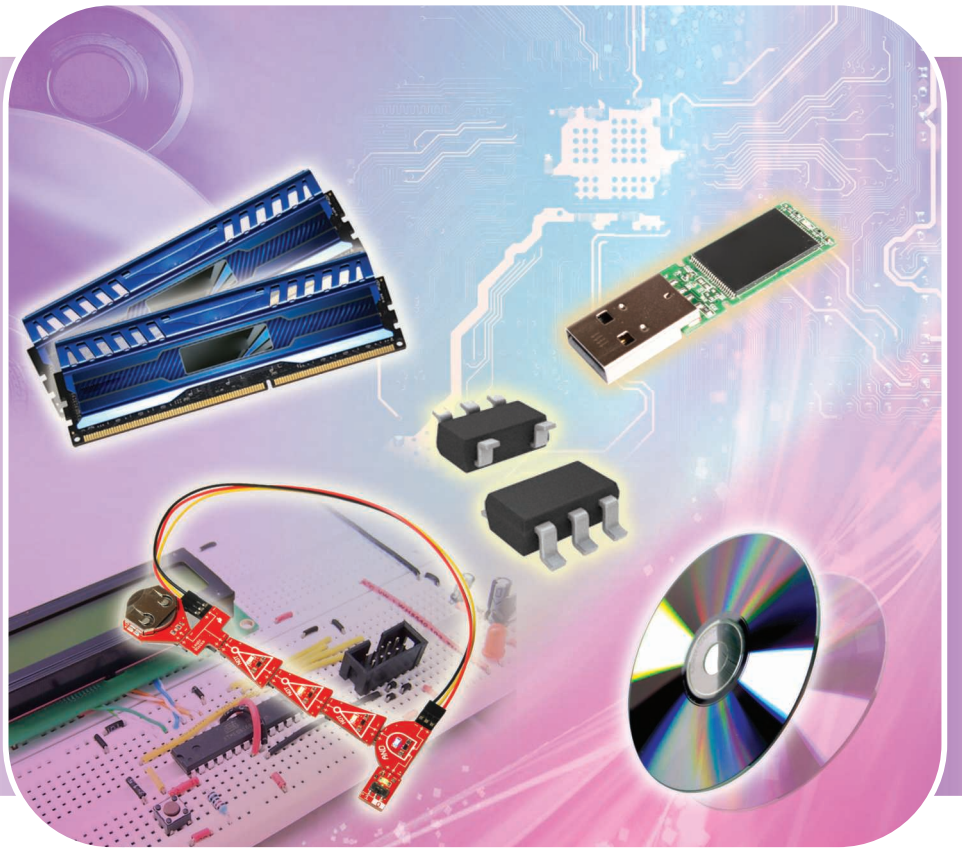
Technical Publications

Digital Electronics

A Conceptual Approach



A. P. Godse, D. A. Godse

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

technical@vtubooks.com
www.vtubooks.com

Digital Electronics

About the Author

A. P. Godse

- Completed M.S in Software Systems with distinction from Birla Institute of Technology.
- Completed B.E. in Industrial Electronics with distinction from University of Pune in 1990.
- Worked as a Professor at Vishwakarma Institute of Technology, Pune.
- Worked as a Technical Director at Noble Institute of Technology, Pune.
- Worked as selection Committee member for M. S. admission for West Virginia University, Washington D.C.
- Developed Microprocessor Based Instruments in co-ordination with Anna Hazare for Environmental Studies Laboratory, at Ralegan Siddhi.
- Developed Microprocessor Lab in-house for Vishwakarma Institute of Technology.
- Worked as Subject Expert for a State Level Technical Paper Presentation Competition, Pune.
- Awarded on 26th Jan 2001 by Pune Municipal Corporation for contributing in education field and technical writing.
- Awarded as a "Parvati Bhushan Puraskar" for contributing in the education field.
- Since 1996, writing books on various engineering subjects. Over the years, many of books are recommended as the reference books and text books in various national and international engineering universities.

D. A. Godse

- Completed M.E and pursuing Ph.D. in Computer Engineering from Bharati Vidyapeeth's University Pune.
- Completed B.E. in Industrial Electronics from University of Pune in 1992.
- Working as a Professor and Head of Information Technology Department in B.V.C.O.E.W, Pune.
- Subject Expert for syllabus setting of Computer Engineering and Information Technology branches at the faculty of Engineering of Pune University.
- Subject Expert and Group Leader for syllabus setting of Electronics, Electronics and Telecommunication and Industrial Electronics branches at the faculty of Maharashtra State, Board of Technical Education.
- Subject In-charge for Laboratory Manual Development, Technical Teacher's Training Institute, Pune.
- Subject In-charge for Question Bank Development Project, Technical Teacher's Training Institute, Pune.
- Subject In-charge for the preparation of Teacher's Guide, Board of Technical Examination, Maharashtra state.
- Subject Expert for a State Level Technical Paper Presentation Competition organized by Bharati Vidyapeeth's Jawaharlal Nehru Institute of Technology, Pune.
- Local Inquiry Committee (LIC) member of Engineering faculty of Pune University.
- Awarded on 15th August 2006 by Pune Municipal Corporation for contributing in education field and technical writing.
- Awarded on the occasion of International Women's Day at Yashawantrao Chavan Pratishthan Sabhagrih, Mumbai by Bharatiya Shikshan Sanstha.

DIGITAL ELECTRONICS

Atul P. Godse

M. S. Software Systems (BITS Pilani)
B.E. Industrial Electronics
Formerly Lecturer in Department of Electronics Engg.
Vishwakarma Institute of Technology
Pune

Mrs. Deepali A. Godse

B.E. Industrial Electronics, M. E. (Computer)
Head of Information Technology Dept.,
Bharati Vidyapeeth's College of Engineering for Women,
Pune

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Pune Nashik Bangalore Chennai Hyderabad
Ahmedabad Bhopal Lucknow Jaipur Delhi

DIGITAL ELECTRONICS

ISBN 9789350992326(Printed Book)

First Edition : January 2014
Reprint : March 2019

© Copyright with Authors

All publishing rights (printed and ebook version) reserved with Technical Publications. No part of this book should be reproduced in any form, Electronic, Mechanical, Photocopy or any information storage and retrieval system without prior permission in writing, from Technical Publications, Pune.

Published by :

TECHNICAL PUBLICATIONS™ Amit Residency, Office No.1, 412, Shaniwar Peth, Pune - 411030, M.S. INDIA.
Ph.: +91-020-24495496/97, Telefax : +91-020-24495497
Email : technical@vtubooks.com Website : www.vtubooks.com

PREFACE

The importance of **Digital Electronics** is well known in various engineering fields. Overwhelming response to our books on various subjects inspired us to write this book. The book is structured to cover the key aspects of the subject **Digital Electronics**.

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

We wish to express our profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by our whole family. We wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Authors
A. P. Godse
D. A. Godse

Dedicated to God.

Table of Contents

Chapter - 1 Minimization Techniques (1 - 1) to (1 - 74)

1.1 Boolean Postulates and Laws ..... 1 - 2

1.1.1 Boolean Algebra Terminology ..... 1 - 2

1.1.2 Fundamental Postulates and Laws of Boolean Algebra ..... 1 - 2

1.1.3 Basic Theorems ..... 1 - 4

1.2 DeMorgan's Theorems ..... 1 - 5

1.3 Principle of Duality ..... 1 - 5

1.4 Boolean Expressions ..... 1 - 9

1.4.1 Sum of Product Form ..... 1 - 10

1.4.2 Product of Sum Form ..... 1 - 11

1.4.3 Standard SOP and Standard POS Forms ..... 1 - 11

1.4.4 Converting Expressions in Standard SOP or POS Form ..... 1 - 12

1.4.4.1 Steps to convert SOP to standard SOP form ..... 1 - 12

1.4.4.2 Steps to convert POS to standard POS form ..... 1 - 14

1.4.5 M Notations : Minterms and Maxterms ..... 1 - 17

1.4.6 Complements of Standard Forms ..... 1 - 19

1.5 Karnaugh Map Minimization ..... 1 - 21

1.5.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps ..... 1 - 21

1.5.2 Plotting a Karnaugh Map ..... 1 - 24

1.5.2.1 Representation of Truth Table on Karnaugh Map ..... 1 - 24

1.5.2.2 Representing Standard SOP on K-Map ..... 1 - 26

1.5.2.3 Representing Standard POS on K-Map ..... 1 - 26

1.5.3 Grouping Cells for Simplification ..... 1 - 28

1.5.3.1 Grouping Two Adjacent Ones (Pair) ..... 1 - 28

1.5.3.2 Grouping Four Adjacent Ones (Quad) . . . . . 1 - 30
1.5.3.3 Grouping Eight Adjacent Ones (Octet) . . . . . 1 - 32
1.5.4 Illegal Grouping . . . . . 1 - 33
1.6 Simplification of SOP Expression . . . . . 1 - 33
1.6.1 Essential Prime Implicants . . . . . 1 - 40
1.6.2 Incompletely Specified Functions (Don't Care Terms) . . . . . 1 - 41
1.6.2.1 Describing Incomplete Boolean Function . . . . . 1 - 41
1.6.2.2 Don't Care Conditions in Logic Design . . . . . 1 - 41
1.6.2.3 Minimization of Incompletely Specified Functions . . . . . 1 - 42
1.7 Simplification of POS Expressions . . . . . 1 - 46
1.8 Five-Variable K-Map . . . . . 1 - 51
1.9 Summary of Rules for K-Map Simplification . . . . . 1 - 55
1.10 Limitations of Karnaugh Map . . . . . 1 - 56
1.11 Quine-McCluskey Method of Minimization . . . . . 1 - 56
1.11.1 Algorithm for Generating Prime Implicants . . . . . 1 - 57
1.11.2 Quine McCluskey using Don't Care Terms . . . . . 1 - 61
1.11.3 Prime Implicant Table and Redundant Prime Implicants . . . . . 1 - 64
1.11.4 Advantages and Disadvantages of Quine McCluskey Method . . . . . 1 - 65
1.12 Two Marks Questions with Answers . . . . . 1 - 69

Chapter - 2 Logic Gates (2 - 1) to (2 - 38)
2.1 Logic Gates . . . . . 2 - 2
2.2 Implementations of Logic Functions using Gates . . . . . 2 - 5
2.2.1 Implementation of SOP Boolean Expression . . . . . 2 - 6
2.2.2 Implementation of POS Boolean Expression . . . . . 2 - 6
2.3 Universal Gates . . . . . 2 - 9
2.3.1 NAND Gate . . . . . 2 - 9
2.3.2 NOR Gate . . . . . 2 - 11
2.4 NAND-NAND Implementation . . . . . 2 - 14

2.5 NOR-NOR Implementation ..... 2 - 21
2.6 Multilevel Gate Implementations ..... 2 - 28
2.7 Multilevel NAND and NOR Implementations ..... 2 - 30
2.8 Multiple Output Implementations ..... 2 - 31
2.9 Two Marks Questions with Answers ..... 2 - 34

Chapter - 3 Logic Families (3 - 1) to (3 - 44)
3.1 Introduction ..... 3 - 2
3.2 Characteristics of Logic Families ..... 3 - 3
3.3 Transistor Transistor Logic (TTL) ..... 3 - 8
3.3.1 TTL Inverter ..... 3 - 9
3.3.2 2-Input TTL NAND Gates ..... 3 - 10
3.3.3 3-Input TTL NAND Gate ..... 3 - 11
3.3.4 Totem-Pole Output ..... 3 - 12
3.3.5 Open-Collector Output ..... 3 - 12
3.3.6 Comparison between Totem-Pole and Open-Collector Outputs ..... 3 - 15
3.3.7 Tri-State TTL Inverter ..... 3 - 15
3.3.8 Characteristics of TTL Family ..... 3 - 17
3.3.9 Advantages and Disadvantages of TTL Family ..... 3 - 19
3.3.10 High-Speed (H) and Low Power (L) TTL ..... 3 - 19
3.3.11 Schottky TTL ..... 3 - 20
3.3.11.1 Schottky (S) and Low-Power Schottky (LS) TTL ..... 3 - 21
3.3.12 Comparison of TTL Series Characteristics ..... 3 - 24
3.3.13 Unconnected Inputs ..... 3 - 25
3.4 CMOS Logic ..... 3 - 26
3.4.1 CMOS Inverter ..... 3 - 26
3.4.2 CMOS NAND Gate ..... 3 - 28
3.4.3 CMOS NOR Gate ..... 3 - 30
3.4.4 Characteristics of CMOS Family ..... 3 - 31
3.4.5 Advantages and Disadvantages of CMOS Family ..... 3 - 33

3.5 Comparison between TTL and CMOS Families ..... 3 - 34

3.6 Interfacing TTL and CMOS Families ..... 3 - 35

3.6.1 TTL Driving CMOS ..... 3 - 36

3.6.2 CMOS Driving TTL ..... 3 - 37

3.7 Two Marks Questions with Answers ..... 3 - 38

Chapter - 4 Combinational Circuits (4 - 1) to (4 - 114)

4.1 Introduction ..... 4 - 2

4.2 Design Procedure ..... 4 - 2

4.3 Design of Adders ..... 4 - 6

4.3.1 Half Adder ..... 4 - 6

4.3.2 Full Adder ..... 4 - 7

4.4 Design of Subtractor ..... 4 - 9

4.4.1 Half Subtractor ..... 4 - 9

4.4.2 Full-Subtractor ..... 4 - 11

4.5 Parallel Adder ..... 4 - 13

4.6 Parallel Subtractor ..... 4 - 14

4.7 Parallel Adder/Subtractor ..... 4 - 15

4.8 Look Ahead-Carry Generator (Fast Adder) ..... 4 - 16

4.9 Four-Bit Parallel Adder (7483/74283) ..... 4 - 20

4.10 Serial Adder/Subtractor ..... 4 - 23

4.11 BCD Adder/Subtractor ..... 4 - 24

4.11.1 BCD Adder ..... 4 - 24

4.11.2 BCD Subtractor ..... 4 - 27

4.11.2.1 9's Complement ..... 4 - 27

4.11.2.2 9's Complement Subtraction ..... 4 - 27

4.11.2.3 10's Complement Subtraction ..... 4 - 28

4.12 Binary Multiplier ..... 4 - 32

4.13 Binary Divider ..... 4 - 39
4.13.1 Restoring Division ..... 4 - 39
4.13.2 Non-Restoring Division ..... 4 - 44
4.13.3 Comparison between Restoring and Non-Restoring Division Algorithm ..... 4 - 49
4.14 Multiplexer ..... 4 - 49
4.14.1 2 : 1 Multiplexer ..... 4 - 50
4.14.2 4 : 1 Multiplexer ..... 4 - 51
4.14.3 8 : 1 Multiplexer ..... 4 - 52
4.14.4 Quadruple 2 to 1 Multiplexer ..... 4 - 53
4.14.5 Expanding Multiplexers ..... 4 - 54
4.14.6 Implementation of Combinational Logic using MUX ..... 4 - 56
4.14.7 Applications of Multiplexer ..... 4 - 66
4.14.8 Multiplexer ICs ..... 4 - 66
4.15 Demultiplexer ..... 4 - 66
4.15.1 Types of Demultiplexers ..... 4 - 67
4.15.1.1 1 : 4 Demultiplexer ..... 4 - 67
4.15.1.2 1 : 8 Demultiplexer ..... 4 - 68
4.15.2 Expanding Demultiplexers ..... 4 - 69
4.15.3 Implementation of Combinational Logic using Demultiplexer ..... 4 - 72
4.15.4 Applications of Demultiplexer ..... 4 - 73
4.15.5 Demultiplexer ICs ..... 4 - 73
4.16 Decoder ..... 4 - 74
4.16.1 Binary Decoder ..... 4 - 74
4.16.2 The 74X138 3-to-8 Decoder ..... 4 - 76
4.16.3 The 74X139 Dual 2 to 4 Decoder ..... 4 - 77
4.16.4 Expanding Cascading Decoders ..... 4 - 77
4.16.5 Realization of Boolean Function using Decoder ..... 4 - 79
4.16.6 Applications of Decoder ..... 4 - 82
4.16.7 Decoder ICs ..... 4 - 82
4.16.8 BCD to Decimal Decoder ..... 4 - 83
4.16.9 BCD to Seven Segment Decoder ..... 4 - 83

4.17 Encoder ..... 4 - 88
4.17.1 Decimal to BCD Encoder ..... 4 - 88
4.17.2 Priority Encoder ..... 4 - 89
4.17.3 Octal to Binary Encoder ..... 4 - 91
4.17.4 Encoder ICs ..... 4 - 91
4.18 Parity Generators and Checkers ..... 4 - 92
4.19 Code Converters ..... 4 - 96
4.20 Magnitude Comparator ..... 4 - 105
4.21 Two Marks Questions with Answers ..... 4 - 109

Chapter - 5 Flip - Flops (5 - 1) to (5 - 40)
5.1 Introduction ..... 5 - 2
5.1.1 Comparison between Combinational and Sequential Logic Circuits ..... 5 - 2
5.1.2 Clock ..... 5 - 3
5.2 One-Bit Memory Cell ..... 5 - 3
5.3 Latches ..... 5 - 5
5.3.1 SR Latch ..... 5 - 5
5.3.2 Gated SR Latch ..... 5 - 6
5.3.3 Gated D Latch ..... 5 - 7
5.4 Flip-Flops ..... 5 - 8
5.4.1 Latches Vs Flip-Flops ..... 5 - 8
5.4.2 Level and Edge Triggering ..... 5 - 8
5.4.3 SR Flip-Flop ..... 5 - 9
5.4.4 D Flip-Flop ..... 5 - 12
5.4.5 JK Flip-Flop ..... 5 - 14
5.4.5.1 JK Flip-Flop using NAND Gates ..... 5 - 17
5.4.5.2 Race-around Condition ..... 5 - 17
5.4.6 Master-Slave SR Flip-Flop ..... 5 - 18
5.4.7 Master-Slave JK Flip-Flop ..... 5 - 19
5.4.8 T Flip-Flop ..... 5 - 20

5.4.9 Asynchronous or Direct Inputs . . . . . 5 - 21

5.5 Various Representation of Flip-Flops . . . . . 5 - 23

5.5.1 Characteristic Equations of Flip-Flops . . . . . 5 - 23

5.5.2 Flip-Flops as Finite State Machines . . . . . 5 - 24

5.5.3 Flip-Flop Excitation Table . . . . . 5 - 24

5.5.3.1 SR Flip-Flop . . . . . 5 - 25

5.5.3.2 JK Flip-Flop . . . . . 5 - 25

5.5.3.3 D Flip-Flop . . . . . 5 - 26

5.5.3.4 T Flip-Flop . . . . . 5 - 26

5.6 Realization of One Flip-Flop using Other Flip-Flop . . . . . 5 - 27

5.6.1 SR Flip-Flop to D Flip-Flop . . . . . 5 - 27

5.6.2 SR Flip-Flop to JK Flip-Flop . . . . . 5 - 28

5.6.3 SR Flip-Flop to T Flip-Flop . . . . . 5 - 29

5.6.4 JK Flip-Flop to T Flip-Flop . . . . . 5 - 30

5.6.5 JK Flip-Flop to D Flip-Flop . . . . . 5 - 31

5.6.6 D Flip-Flop to T Flip-Flop . . . . . 5 - 31

5.6.7 T Flip-Flop to D Flip-Flop . . . . . 5 - 33

5.6.8 JK Flip-Flop to SR Flip-Flop . . . . . 5 - 33

5.6.9 D Flip-Flop to SR Flip-Flop . . . . . 5 - 34

5.6.10 T Flip-Flop to SR Flip-Flop . . . . . 5 - 35

5.6.11 D Flip-Flop to JK Flip-Flop . . . . . 5 - 35

5.7 Two Marks Questions with Answers . . . . . 5 - 37

Chapter - 6 Counters (6 - 1) to (6 - 42)

6.1 Introduction . . . . . 6 - 2

6.2 Ripple / Asynchronous Counters . . . . . 6 - 4

6.2.1 Asynchronous / Ripple Down Counter . . . . . 6 - 6

6.2.2 Asynchronous Up / Down Counter . . . . . 6 - 8

6.2.3 Decoding Gates . . . . . 6 - 9

6.2.4 Problem Faced by Ripple Counters (Glitch Problem) . . . . . 6 - 11

6.3 Design of Ripple (Asynchronous) Counters . . . . . 6 - 12

6.4 Synchronous Counters ..... 6 - 15
6.4.1 2-bit Synchronous Binary Up Counter ..... 6 - 16
6.4.2 3-bit Synchronous Binary Up Counter ..... 6 - 16
6.4.3 4-bit Synchronous Binary Up Counter ..... 6 - 17
6.4.4 Synchronous Down and Up/Down Counters ..... 6 - 18
6.4.5 Synchronous Vs Asynchronous Counters ..... 6 - 19
6.5 Design of Synchronous Counters ..... 6 - 20
6.6 Two Marks Questions with Answers ..... 6 - 42

Chapter - 7 Shift Registers (7 - 1) to (7 - 28)
7.1 Introduction ..... 7 - 2
7.1.1 Buffer Register ..... 7 - 2
7.1.2 Controlled Buffer Register ..... 7 - 2
7.2 Shift Registers ..... 7 - 4
7.3 Types of Shift Registers ..... 7 - 4
7.3.1 Serial In Serial Out (SISO) Shift Register ..... 7 - 4
7.3.2 Serial In Parallel Out (SIPO) Shift Register ..... 7 - 8
7.3.3 Parallel In Serial Out (PISO) Shift Register ..... 7 - 9
7.3.4 Parallel In Parallel Out (PIPO) Shift Register ..... 7 - 9
7.3.5 Bidirectional Shift Register ..... 7 - 10
7.4 Universal Shift Register ..... 7 - 12
7.5 Applications of Shift Registers ..... 7 - 13
7.5.1 Delay Line ..... 7 - 13
7.5.2 Serial-to-Parallel Converter ..... 7 - 14
7.5.3 Parallel-to-Serial Converter ..... 7 - 14
7.5.4 Shift Register Counters ..... 7 - 14
7.5.5 Pseudo-Random Binary Sequence (PRBS) Generator ..... 7 - 14
7.5.6 Sequence Generator ..... 7 - 14
7.5.7 Sequence Detector ..... 7 - 15
7.6 Ring Counters ..... 7 - 15

7.7 Johnson or Twisting Ring or Switch Tail Counter ..... 7 - 20

7.8 Two Marks Questions with Answers ..... 7 - 27

Chapter - 8 Analysis and Design of Synchronous Sequential Circuits (8 - 1) to (8 - 110)

8.1 Introduction ..... 8 - 2

8.2 Clocked Sequential Circuits ..... 8 - 2

8.2.1 Moore Model ..... 8 - 3

8.2.2 Mealy Model ..... 8 - 4

8.2.3 Moore vs Mealy Circuit Models ..... 8 - 5

8.2.4 Representations of Sequential Circuits ..... 8 - 5

8.2.4.1 State Diagram ..... 8 - 5

8.2.4.2 State Table ..... 8 - 6

8.2.4.3 Transition Table ..... 8 - 6

8.2.5 Conversion of Models ..... 8 - 7

8.3 Analysis of Clocked Sequential Circuits ..... 8 - 9

8.4 Design of Clocked Sequential Circuits ..... 8 - 20

8.4.1 State Reduction ..... 8 - 21

8.4.2 Reduction of State Table using Partition Technique ..... 8 - 26

8.4.3 Reduction of State Table using Merger Graph ..... 8 - 29

8.4.4 Reduction of State Table using Merger Table ..... 8 - 32

8.4.5 State Assignment ..... 8 - 34

8.4.6 Choice of Flip-Flops and Derivation of Next State and Output ..... 8 - 35

8.4.7 State Assignment Problem ..... 8 - 42

8.4.8 Design with Unused States ..... 8 - 46

8.4.9 Lockout Condition ..... 8 - 51

8.5 Sequence Generator ..... 8 - 65

8.5.1 Sequence Generator using Counters ..... 8 - 65

8.5.2 Sequence Generator using Shift Register ..... 8 - 80

8.6 Sequence Detector ..... 8 - 89

8.7 Simplification of Incompletely Specified Machines ..... 8 - 99

8.8 Algorithmic State Machine ..... 8 - 100
8.8.1 ASM Symbols/Notations ..... 8 - 101
8.8.2 Relation between ASM Chart and State Diagram ..... 8 - 102
8.9 Two Marks Questions with Answers..... 8 - 109

Chapter - 9 Asynchronous Sequential Circuits (9 - 1) to (9 - 64)
9.1 Introduction ..... 9 - 2
9.2 Types of Asynchronous Circuits ..... 9 - 2
9.3 Analysis of Pulse Mode Asynchronous Circuits ..... 9 - 4
9.4 Design of Pulse Mode Circuit ..... 9 - 10
9.5 Analysis of Fundamental Mode Sequential Circuits ..... 9 - 16
9.6 Design of Fundamental Mode Sequential Circuits..... 9 - 23
9.6.1 Derivation of Primitive Flow Table ..... 9 - 23
9.6.2 Reduction of Primitive Flow Table..... 9 - 25
9.6.3 Race Free State Assignment..... 9 - 28
9.6.3.1 Races and Cycles ..... 9 - 29
9.6.3.2 Shared Row State Assignment ..... 9 - 31
9.6.3.3 One Hot State Assignment ..... 9 - 31
9.6.4 Realization of Flow Table ..... 9 - 32
9.7 Data Synchronizers ..... 9 - 50
9.8 Mixed Operating Mode Asynchronous Circuits ..... 9 - 50
9.9 Design of Hazard Free Switching Circuits..... 9 - 52
9.9.1 Eliminating a Hazard..... 9 - 53
9.9.2 Hazards in Sequential Circuits ..... 9 - 54
9.9.3 Essential Hazards ..... 9 - 56
9.9.4 Eliminating Essential Hazards ..... 9 - 56
9.10 Two Marks Questions with Answers..... 9 - 61

Chapter - 10 Memory Devices and PLDs (10 - 1) to (10 - 82)
10.1 Introduction ..... 10 - 2

10.2 Classification of Memory ..... 10 - 3
10.3 ROM (Read Only Memory) ..... 10 - 4
10.3.1 PROM (Programmable Read Only Memory) ..... 10 - 5
10.3.2 EPROM (Erasable Programmable Read Only Memory) ..... 10 - 6
10.3.3 EEPROM (Electrically Erasable Programmable Read Only Memory) ..... 10 - 7
10.4 ROM Organization ..... 10 - 8
10.5 RAM (Random Access Memories) ..... 10 - 10
10.5.1 Static RAM (SRAM) ..... 10 - 10
10.5.1.1 Static RAM Cell ..... 10 - 10
10.5.1.2 Bipolar RAM Cell ..... 10 - 11
10.5.1.3 MOSFET RAM Cell ..... 10 - 11
10.5.2 Dynamic RAM (DRAM) ..... 10 - 12
10.5.2.1 Dynamic RAM Cell ..... 10 - 12
10.5.2.2 Comparison between SRAM and DRAM ..... 10 - 12
10.6 RAM Organization ..... 10 - 13
10.7 DRAM Organization ..... 10 - 14
10.8 Memory Cycles and Timing Waveforms ..... 10 - 15
10.9 Memory Decoding ..... 10 - 18
10.10 Memory Expansion ..... 10 - 22
10.10.1 Expanding Word Size ..... 10 - 22
10.10.2 Expanding Memory Capacity ..... 10 - 23
10.10.3 Limitations for Memory Expansion ..... 10 - 24
10.11 Programmable Logic Devices (PLDs) ..... 10 - 25
10.11.1 PROM (Programmable Read Only Memory) ..... 10 - 26
10.11.1.1 AND Matrix ..... 10 - 29
10.11.1.2 OR Matrix ..... 10 - 30
10.11.1.3 Invert / Non-invert Matrix ..... 10 - 32
10.11.1.4 Combinational Logic Implementation using PROM ..... 10 - 32
10.11.2 PLA (Programmable Logic Array) ..... 10 - 39

10.11.2.1 Input Buffer . . . . . 10 - 40
10.11.2.2 Output Buffer . . . . . 10 - 40
10.11.2.3 Output through Flip-Flops . . . . . 10 - 40
10.11.2.4 Implementation of Combination Logic Circuit using PLA . . . . . 10 - 40
10.11.3 PAL (Programmable Array Logic) . . . . . 10 - 55
10.11.3.1 Implementation of Combinational Logic Circuit using PAL . . . . . 10 - 56
10.11.4 Comparison between PROM, PLA and PAL . . . . . 10 - 63
10.11.5 Registered PALs . . . . . 10 - 63
10.11.6 Configurable PALs . . . . . 10 - 67
10.11.7 EX-OR PALs . . . . . 10 - 74
10.12 Field Programmable Gate Arrays (FPGA) . . . . . 10 - 76
10.12.1 Applications of FPGA . . . . . 10 - 77
10.12.2 SRAM Based FPGA . . . . . 10 - 78
10.13 Two Marks Questions and Answers . . . . . 10 - 79

Chapter - 11 Design of Combinational and Sequential Circuits using VERILOG (11 - 1) to (11 - 36)

11.1 Introduction . . . . . 11 - 2
11.2 Structure of Verilog Module . . . . . 11 - 2
11.3 Operators in Verilog HDL . . . . . 11 - 4
11.4 Verilog Data Types . . . . . 11 - 7
11.5 HDL for Combinational Circuits . . . . . 11 - 8
11.5.1 Structure of the Data Flow Description . . . . . 11 - 9
11.5.2 Structure of the Behavioral Description . . . . . 11 - 14
11.5.3 Gate Level/Structural Description . . . . . 11 - 18
11.6 HDL for Sequential Circuits . . . . . 11 - 22
11.6.1 Description of D-Latch . . . . . 11 - 22
11.6.2 Description of Flip-Flops . . . . . 11 - 23
11.6.3 Description of Sequential Circuits . . . . . 11 - 25
11.6.4 HDL for Registers and Counters . . . . . 11 - 28

Appendix A Datasheets for Commonly used Digital ICs (A - 1) to (A - 32)

1 Minimization Techniques

Syllabus

Boolean postulates and laws - De-Morgan's theorem - Principle of duality - Boolean expression - Minimization of Boolean expressions - Minterm - Maxterm - Sum of Products (SOP) - Product of Sums (POS) - Karnaugh map minimization - Don't care conditions - Quine-McCluskey method of minimization.

Contents

1.1 Boolean Postulates and Laws **May-08, 11, Dec.-08**

1.2 DeMorgan's Theorems

1.3 Principle of Duality **May-03, 04, 05, 08, 10, 11, 12,**
. . . . . **Dec.-03, 05, 07, 08, 11, 12** · · Marks 8

1.4 Boolean Expressions **Dec.-03, 07, 08, 09**
. . . . . **May-08, 10,12** · · · · · · · · · · · · Marks 7

1.5 Karnaugh Map Minimization

1.6 Simplification of SOP Expression **Dec.-03, 05,**
. . . . . **May-05, 06, 09, 10, 11** · · · · Marks 12

1.7 Simplification of POS Expressions . . . . . **May-04, 09** · · · · · · · · · · · · · Marks 10

1.8 Five-Variable K-Map . . . . . **Dec.-03, 11, May-08** · · · · · · · Marks 8

1.9 Summary of Rules for K-Map Simplification

1.10 Limitations of Karnaugh Map

1.11 Quine-McCluskey Method of Minimization **Dec.-06, 09, 08, 10, 12,**
. . . . . **May-07, 09, 10, 12** · · · · · · · Marks 16

1.12 Two Marks Questions with Answers

1.1 Boolean Postulates and Laws

May-08, 11, Dec.-08

In 1854, George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called **Boolean algebra.**

Boolean algebra is a system of mathematical logic. It differs from both ordinary algebra and the binary number system.

1.1.1 Boolean Algebra Terminology

Variable : The symbol which represent an arbitrary elements of an Boolean algebra is known as **variable**. Any single variable or a function of several variables can have either a 1 or 0 value. For example, in expression Y = A + BC, variables A, B and C can have either a 1 or 0 value, and function Y also can have either a 1 or 0 value; however its value depends on the value of Boolean expression.

Constant : In expression Y = A + 1, the first term A is a variable and have value either a 1 or 0. The second term has a fixed value 1. So 1 is a constant here. The constant term may be 0 or 1.

Complement : A complement of a variable is represented by a "bar" over the letter. For example, the complement of a variable A will be denoted by $\overline{A}$. So if A = 1, $\overline{A}$ = 0 and if A = 0, $\overline{A}$ = 1. Sometimes a prime symbol (′) is used to denote the complement. For example, the complement of A can be written as A′.

Literal : Each occurrence of a variable in Boolean function either in a complemented or an uncomplemented form is called a **literal**.

Boolean function : Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as **Boolean formulas**. We use Boolean expressions to describe **Boolean functions**. For example, if the Boolean expression (A + $\overline{B}$) C is used to describe the function f, then Boolean function is written as

f(A, B, C) = (A + $\overline{B}$) C or f = (A + $\overline{B}$) C

1.1.2 Fundamental Postulates and Laws of Boolean Algebra

The postulates of a mathematical system form the basic assumption from which it is possible to deduce the theorems, laws and properties of the system. Boolean algebra is formulated by a defined set of elements, together with two binary operators, + and · , provided that the given postulates are satisfied.

- **Closure (a) :** Closure with respect to the operator + : *When two binary elements are operated by operator + the result is a unique binary element.*

- **Closure (b) :** Closure with respect to the operator · (dot) : *When two binary elements are operated by operator · (dot), the result is a unique binary element.*
- An identity element with respect to +, designated by 0 : **A + 0 = 0 + A = A**
- An identity element with respect to · , designated by 1 : **A · 1 = 1 · A = A**
- Commutative with respect to + : **A + B = B + A**
- Commutative with respect to · : **A · B = B · A**
- Distributive property of · over + : **A · (B + C) = (A · B) + (A · C)**
- Distributive property of + over · : **A + (B · C) = (A + B) · (A + C)**
- Associative property of + : **A + (B + C) = (A + B) + C**
- Associative property of · : **(A · B) · C = A · (B · C)**
- For every binary element, there exists complement element. For example, if A is an element, we have $\overline{A}$ is a complement of A. i.e. if A = 0, $\overline{A}$ = 1 and if A = 1, $\overline{A}$ = 0.
- There exists at least two elements, say A and B in the set of binary elements such that A ≠ B.

From the above discussion we can summarize the postulates of Boolean algebra as shown in Table. 1.1.1.

Sr. No.	Postulate	Comment
1.	Result of each operation is either 0 or 1	1, 0 ∈ B
2.	a) 0 + 0 = 0 0 + 1 = 1 + 0 = 1	**Identity elements**
	b) 1 · 1 = 1 1 · 0 = 0 · 1 = 0	
3.	a) (A + B) = (B + A)	**Commutative law 1** : The order in which the variables are ORed makes no difference in the output.
	b) (A · B) = (B · A)	**Commutative law 2 :** The order in which the variables are ANDed makes no difference in the output.
4.	a) A · (B + C) = (A · B) + (A · C)	**Distributed law 1 :** It states that ORing several variables and ANDing the result with a single variable is equivalent to ANDing the result with a single variable with each of the several variables and then ORing the products.
	b) A + (B · C) = (A + B) · (A + C)	**Distributed law 2** : It states that ANDing several variables and ORing the result with a single variable is equivalent to ORing the result with a single variable with each of the several variables and then ANDing the sums.

5.	a) $A + \overline{A} = 1$ since $0 + \overline{0} = 0 + 1 = 1$ and $1 + \overline{1} = 1 + 0 = 1$ b) $A \cdot \overline{A} = 0$ since $0 \cdot \overline{0} = 0 \cdot 1 = 0$ and $1 \cdot \overline{1} = 1 \cdot 0 = 0$	**Complement**
6.	$A + (B + C) = (A + B) + C$ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$	**Associative Law 1 :** It states that in the ORing of several variables, the result is the same regardless of the grouping of the variables. **Associative Law 2 :** It states that it makes no difference in what order the variables are grouped when ANDing several variables.

Table 1.1.1 Fundamental postulates of Boolean algebra

1.1.3 Basic Theorems

The Table 1.1.2 lists the five basic theorems of Boolean algebra four of its postulates. The postulates and theorems are listed in pairs and designated by part (a) and part (b). One part may be obtained from the other by using principle of duality.

Postulates	**(a)**	**(b)**
Postulate 2 (Identity)	$A + 0 = A$	$A \cdot 1 = A$
Postulate 5 (Complement)	$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
Postulate 3 (Commutative)	$A + B = B + A$	$AB = BA$
Postulate 4 (Distributive)	$A (B + C) = AB + AC$	$A + BC = (A + B) (A + C)$
Theorems	**(a)**	**(b)**
Theorem 1 (Idempotency)	$A + A = A$	$A \cdot A = A$
Theorem 2	$A + 1 = 1$	$A \cdot 0 = 0$
Theorem 3 (Involution)	$\overline{\overline{A}} = A$	
Theorem 4 (Absorption)	$A + AB = A$	$A (A + B) = A$
Theorem 5	$A + \overline{A}B = A + B$	$A \cdot (\overline{A} + B) = AB$
Theorem 6 (Associative)	$A + (B + C) = (A + B) + C$	$A (BC) = (AB) C$

Table 1.1.2 Postulates and basic theorems of Boolean algebra

1.2 DeMorgan's Theorems

DeMorgan suggested two theorems that form an important part of Boolean algebra. In the equation form, they are : 1. $\overline{AB} = \overline{A} + \overline{B}$ 2. $\overline{A + B} = \overline{A} \cdot \overline{B}$

1. $\overline{AB} = \overline{A} + \overline{B}$ **:** The complement of a product is equal to the sum of the complements. This is illustrated by truth Table 1.2.1.

A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Table 1.2.1 Truth table

2. $\overline{A + B} = \overline{A} \cdot \overline{B}$ **:** The complement of a sum is equal to the product of the complements. The truth Table 1.2.2 illustrates this law.

A	B	$\overline{A + B}$	$\overline{A}\ \overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Table 1.2.2 Truth table

1.3 Principle of Duality

May-03, 04, 05, 08, 10, 11, 12, Dec.-03, 05, 07, 08, 11, 12

The principle of duality theorem says that, starting with a Boolean relation, you can derive another Boolean relation by

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign and
3. Complementing any 0 or 1 appearing in the expression.

For example : Dual of relation $A + \overline{A} = 1$ is $A \cdot \overline{A} = 0$

Illustrative Examples

Example 1.3.1 *Simplify : x + x'y.* **Dec.-07, May-10, Marks 2**

Solution :

$$z = x + \overline{x}y = x + xy + \overline{x}y \qquad \because\ A + AB = A$$
$$= x + y\,(x + \overline{x}) \qquad \because\ x + \overline{x} = 1$$
$$= x + y$$

Example 1.3.2 *Apply DeMorgans theorem to simplify* $\overline{A + B\overline{C}}$ **May-04**

Solution :

$$\overline{A + B\overline{C}} = \overline{A} \cdot \overline{B\overline{C}}$$
$$= \overline{A} \cdot (\overline{B} + C) = \overline{A}\,\overline{B} + \overline{A}\,C$$

Example 1.3.3 *Prove the following Boolean identities.*

$(x_1 + x_2)(\bar{x}_1\bar{x}_3 + x_3)\overline{(\bar{x}_2 + x_1x_3)} = \bar{x}_1x_2$. **May-03, May-08, Marks 8**

Solution :

$(x_1 + x_2)(\bar{x}_1\bar{x}_3 + x_3)\overline{(\bar{x}_2 + x_1 x_3)} = \bar{x}_1x_2$

L.H.S. $= (x_1 + x_2)(\bar{x}_1\bar{x}_3 + x_3)(\bar{\bar{x}}_2 \cdot \overline{x_1x_3})$

$= (x_1 + x_2)(\bar{x}_1\bar{x}_3 + x_3)(x_2 \cdot (\bar{x}_1 + \bar{x}_3))$

$= (x_1 + x_2)(\bar{x}_1\bar{x}_3 + x_3)(x_2\bar{x}_1 + x_2\bar{x}_3)$

$= (x_1 + x_2)(\bar{x}_1 + x_3)(x_2\bar{x}_1 + x_2\bar{x}_3)$ $\because A + \overline{A}B = A + B$

$= (0 + x_1x_3 + \bar{x}_1x_2 + x_2x_3)(x_2\bar{x}_1 + x_2\bar{x}_3)$ $\because A\overline{A} = 0$

$= 0 + 0 + \bar{x}_1x_2 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3 + 0$

$= \bar{x}_1x_2 + \bar{x}_1x_2\bar{x}_3 + \bar{x}_1x_2x_3$

$= \bar{x}_1x_2 (1 + \bar{x}_3 + x_3)$

$= \bar{x}_1x_2$ … R.H.S. $1 + A = 1$

Example 1.3.4 $\overline{A}BC\overline{D} + BC\overline{D} + B\overline{C}\,\overline{D} + B\,\overline{C}\,D$

Solution :

$= BC\overline{D}(\overline{A} + 1) + B\overline{C}\,\overline{D} + B\overline{C}\,D$ Distributive

$= BC\overline{D} + B\overline{C}\,\overline{D} + B\overline{C}\,D$ Theorem 2(a) : $[A + 1 = 1]$

$= B\overline{D}(C + \overline{C}) + B\overline{C}\,D$ Distributive

$= B\overline{D} + B\overline{C}\,D$ Postulate 5(a) : $[A + \overline{A} = 1]$

$= B(\overline{D} + \overline{C}\,D)$ Distributive

$= B(\overline{D} + \overline{C})$ Theorem 5(a) : $[A + \overline{A}B = A + B]$

Example 1.3.5 $AB + \overline{AC} + A\overline{B}\,C\,(AB + C)$

$= AB + \overline{AC} + AA\overline{B}\,BC + A\overline{B}\,CC$ Distributive

$= AB + \overline{AC} + A\overline{B}\,CC$ Postulate 5(b) : $[A \cdot \overline{A} = 0]$

$= AB + \overline{AC} + A\overline{B}\,C$ Theorem 1(b) : $[A \cdot A = A]$

$= AB + \overline{A} + \overline{C} + A\,\overline{B}\,C$ DeMorgan's Theorem 1 : $[\overline{AB} = \overline{A} + \overline{B}]$

$= \overline{A} + B + \overline{C} + A\overline{B}\,C$ Theorem 5(a) : $[A + \overline{A}B = A + B]$

$= \overline{A} + A\overline{B}\,C + B + \overline{C}$ Commutative

$= \overline{A} + \overline{B}C + B + \overline{C}$ Theorem 5(a) : $[A + \overline{A}B = A + B]$ Here $B = \overline{B}C$

$= \overline{A} + B + \overline{C} + \overline{B}C$ Commutative

$= \overline{A} + B + \overline{C} + \overline{B}$ Theorem 5(a) : $[A + \overline{A}B = A + B]$

$= \overline{A} + \overline{C} + 1$ Postulate 5(a) : $[A + \overline{A} = 1]$

$= 1$ Theorem 2(a) : $[A + 1 = 1]$

Examples with Solutions

Example 1.3.6 *If A and B are Boolean variables and if A = 1 and* $\overline{A+B} = 0$*, find B.* **Dec.-03, Marks 2**

Solution : Given $\overline{A+B} = 0$

$\therefore$ $A + B = 1$, Now if A = 1 we have,

$1 + B = 1$ and therefore B can be either 0 or 1.

Example 1.3.7 *Simplify* $A + AB + \overline{A} + B$. **May-05, Marks 2**

Solution : $A + AB + \overline{A} + B = A(1 + B) + \overline{A} + B = A + \overline{A} + B = 1 + B = \mathbf{1}$.

Example 1.3.8 *Apply DeMorgan's theorem to the following expression.*
((A + B + C) D)' **Dec.-07, Marks 4**

Solution : $\overline{((A+B+C)D)} = \overline{(A+B+C)} + \overline{D} = \mathbf{\overline{A}\,\overline{B}\,\overline{C} + \overline{D}}$

Example 1.3.9 *Using Boolean laws and rules simplify the logic expression.*
Z = (A' + B) (A + B) **Dec.-07, Marks 4**

Solution : $Z = (\overline{A} + B)(A + B) = A\overline{A} + \overline{A}B + AB + B = 0 + B(\overline{A} + A + 1) = \mathbf{B}$

Example 1.3.10 *Prove the following using DeMorgan's theorem*
$AB + CD = \overline{\overline{AB} \cdot \overline{CD}}$ *and* $(A+B)\cdot(C+D) = \overline{\overline{A+B} + \overline{C+D}}$. **May-08, Marks 4**

Solution :

1) $AB + CD = \overline{\overline{AB + CD}}$ $\because \overline{\overline{Y}} = Y$

$= \overline{\overline{AB} \cdot \overline{CD}}$ $\because \overline{A+B} = \overline{A} \cdot \overline{B}$

2) $(A+B)\cdot(C+D) = \overline{\overline{(A+B)\cdot(C+D)}}$ $\because \overline{\overline{Y}} = Y$

$= \overline{\overline{(A+B)} + \overline{(C+D)}}$ $\because \overline{A \cdot B} = \overline{A} + \overline{B}$

Example 1.3.11 *Apply DeMorgan's theorem for the function* $\overline{(A+B+C)D}$. **Dec.-08, Marks 3**

Solution : $\overline{(A+B+C)\,D} = \overline{(A+B+C)} + \overline{D} = \overline{A}\cdot\overline{B}\cdot\overline{C} + \overline{D} = \overline{A}\,\overline{B}\,\overline{C} + \overline{D}$

Example 1.3.12 *Find the complement of A + BC + AB.* **Dec.-08, Marks 3**

Solution :

$$\begin{aligned} F &= \overline{A+BC+AB} = \overline{A+AB+BC} \\ &= \overline{A+BC} \qquad\qquad A+AB = A \\ &= \overline{A}\cdot\overline{BC} = \overline{A}\cdot(\overline{B}+\overline{C}) = \overline{A}\,\overline{B} + \overline{A}\,\overline{C} \end{aligned}$$

Example 1.3.13 *Simplify the following Boolean expression to a minimum number of literals* $(BC'+A'D)(AB'+CD')$. **Dec.-11, Marks 2**

Solution :

$$\begin{aligned} Y &= (B\overline{C} + \overline{A}D)\ (A\overline{B} + C\overline{D}) = AB\overline{B}\,\overline{C} + BC\overline{C}\,\overline{D} + A\overline{A}\,\overline{B}D + \overline{A}CD\overline{D} \\ &= \mathbf{0} \qquad\qquad \because\ A\overline{A} = 0 \end{aligned}$$

Example 1.3.14 *Simplify the Boolean expression.*
$F = x'y'z' + x'yz + xy'z' + xyz'$. **May-12, Marks 5**

Solution :

$$\begin{aligned} F &= x'y'z' + x'yz + xy'z' + xyz' \\ &= \overline{x}\,\overline{y}\,\overline{z} + x\,\overline{y}\,\overline{z} + \overline{x}\,y\,z + x\,\overline{y}\,\overline{z} + x\,y\,\overline{z} \qquad \because\ A + A = A \\ &= \overline{y}\,\overline{z}\,(\overline{x}+x) + \overline{x}\,y\,z + x\,\overline{z}\,(\overline{y}+y) \\ &= \overline{y}\,\overline{z} + \overline{x}\,y\,z + x\,\overline{z} \qquad \because\ A + \overline{A} = 1 \end{aligned}$$

Example 1.3.15 *Simplify the given Boolean expression* $F = x' + xy + xz' + xy'z'$. **Dec.-12, Marks 2**

Solution :

$$\begin{aligned} F &= \overline{x} + x\,y + x\,\overline{z} + x\,\overline{y}\,\overline{z} \\ &= \overline{x} + y + \overline{z} + \overline{y}\,\overline{z} \qquad \because\ A + \overline{A}\,B = A+B \\ &= \overline{x} + y + \overline{z}(1+\overline{y}) \\ &= \overline{x} + y + \overline{z} \qquad \because\ 1+\overline{A} = 1 \end{aligned}$$

Examples for Practice

Example 1.3.16 *Simplify the following Boolean expressions to a minimum number of literals :*
$XY + \overline{X}Z + YZ$ **Dec.-05, Marks 2**

Ans. : XY + $\overline{X}$Z

Example 1.3.17 *Simplify the following Boolean expression.*
$a\bar{b}\,\bar{c} + a\bar{b}c + abc.$

May-05, Marks 2

Ans. : $a\bar{b}$ + ac

Example 1.3.18 *Simplify the following Boolean expressions to a minimum number of literals :*
$(X+Y)\ \left(X+\bar{Y}\right)$

Dec.-05, Marks 2

Ans. : X

Review Questions

1. *What are Boolean variables ?*
2. *Define the following terms : Boolean variable, complement, literal.*
3. *State the fundamental postulates of Boolean algebra.*
4. *State various laws of Boolean algebra.*
5. *State the associative law of Boolean algebra.* **May-08, Marks 2**
6. *State and prove DeMorgan's theorem.* **May-11, 12, Dec.-08, Marks 8**
7. *Explain the principle of duality with the help of example.*

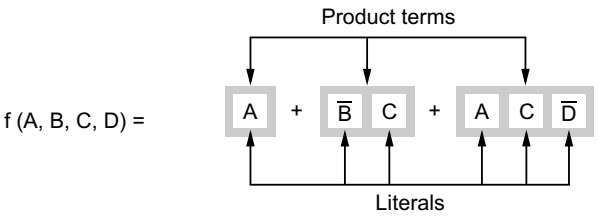
1.4 Boolean Expressions

Dec.-03, 07, 08, 09, May-08, 10,12

Boolean expressions are constructed by connecting the Boolean constants and variables with the Boolean operations. These Boolean expressions are also known as **Boolean formulas**. We use Boolean expressions to describe switching function or **Boolean functions**. For example, if the Boolean expression (A + $\bar{B}$) C is used to describe the function f, then Boolean function is written as

f(A, B, C) = (A + $\bar{B}$) C or f = (A + $\bar{B}$) C

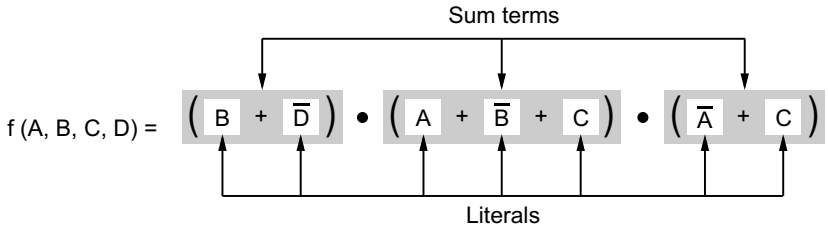
Based on the structure of Boolean expression, it can be categorized in different formulas. One such categorization are the normal formulas. Let us consider the four-variable Boolean function.



... (1.4.1)

In this Boolean function the variables are appeared either in a complemented or an uncomplemented form. Each occurrence of a variable in either a complemented or an uncomplemented form is called a **literal**. Thus, the above Boolean function 1.4.1 consists of six literals. They appear in the product terms. A **product term** is defined as either a

literal or a product (also called conjunction) of literals. Function 1.4.1 contains three product terms, namely, A, $\overline{B}$ C and A C $\overline{D}$. Let us consider another four variable Boolean function.



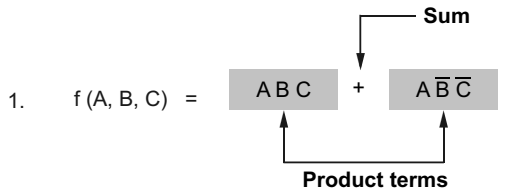
... (1.4.2)

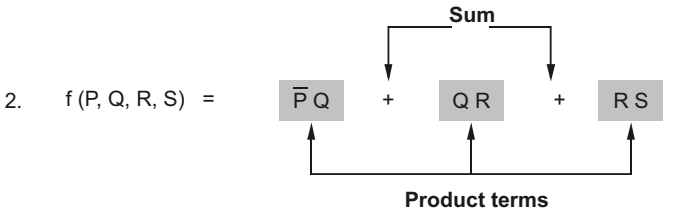
The above Boolean function consists of seven literals. Here, they appear in the sum terms. A **sum term** is defined as either a literal or a sum (also called disjunction) of literals. Function 1.4.2 contains three sum terms, namely, (B + $\overline{D}$), (A + $\overline{B}$ + C) and ($\overline{A}$ + C). These literals and terms are arranged in one of the two forms :

- Sum of product form (SOP) and
- Product of sum form (POS).

1.4.1 Sum of Product Form

The words sum and product are derived from the symbolic representations of the OR and AND functions by + and · (addition and multiplication), respectively. But we realize that these are not arithmetic operators in the usual sense. A product term is any group of literals that are ANDed together. For example, ABC, XY and so on. A sum term is any group of literals that are ORed together such as A + B + C, X + Y and so on. A sum of products (SOP) is a group of product terms ORed together. Some examples of this form are :

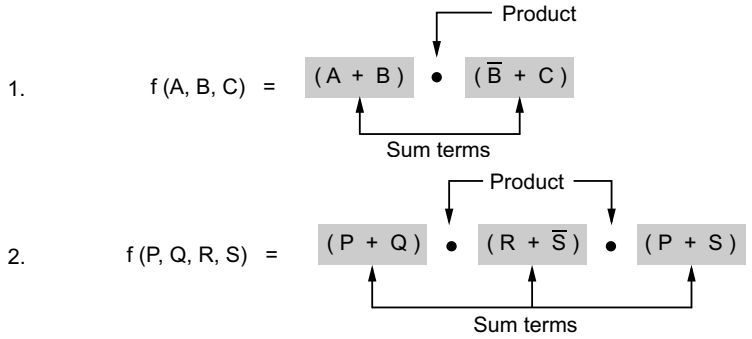




Each of these sum of products expressions consist of two or more product terms (AND) that are ORed together. Each product term consists of one or more literals appearing in either complemented or uncomplemented form. For example, in the sum of products expression $ABC + A\overline{B}\overline{C}$, the first product term contains literals A, B and C in their uncomplemented form. The second product term contains B and C in their complemented (inverted) form. The sum of product form is also known as **disjunctive normal form** or **disjunctive normal formula**.

1.4.2 Product of Sum Form

A product of sums is any groups of sum terms ANDed together. Some examples of this form are :



Each of these product of sums expressions consist of two or more sum terms (OR) that are ANDed together. Each sum term consists of one or more literals appearing in either complemented or uncomplemented form. The product of sum form is also known as **conjunctive normal form** or **conjunctive normal formula**.

1.4.3 Standard SOP and Standard POS Forms

We can realize that in the SOP form, all the individual terms do not involve all literals. For example, in expression $AB + A\overline{B}C$ the first product term do not contain literal C. If each term in SOP form contains all the literals then the SOP form is known as **standard or canonical SOP form**. Each individual term in the standard SOP form is called **minterm**. One standard sum of products expression is as shown in Fig. 1.4.1.

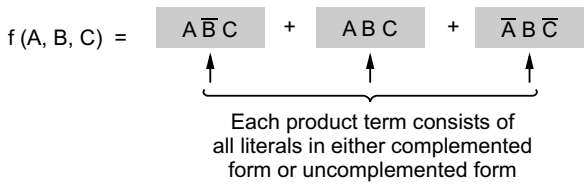


Fig. 1.4.1 Standard SOP form

If each term in POS form contains all the literals then the POS form is known as **standard or canonical POS form**. Each individual term in the standard POS form is called **maxterm**. One standard product of sums expression is as shown in Fig. 1.4.2.

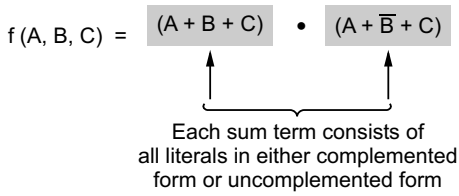


Fig. 1.4.2 Standard POS form

1.4.4 Converting Expressions in Standard SOP or POS Form

Sum of product form can be converted to standard sum of products by ANDing the terms in the expression with terms formed by ORing the literal and its complement which are not present in that term. For example for a three literal expression with literals A, B and C, if there is a term AB, where C is missing, then we form term $(C + \overline{C})$ and AND it with AB. Therefore, we get $AB\ (C + \overline{C}) = ABC + AB\overline{C}$.

1.4.4.1 Steps to convert SOP to standard SOP form

Step 1 : Find the missing literal in each product term if any.

Step 2 : AND each product term having missing literal/s with term/s form by ORing the literal and its complement.

Step 3 : Expand the terms by applying distributive law and reorder the literals in the product terms.

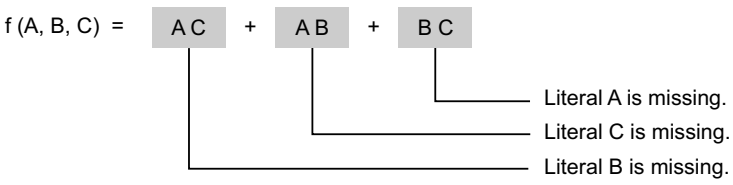
Step 4 : Reduce the expression by omitting repeated product terms if any. Because A + A = A.

Examples for Understanding

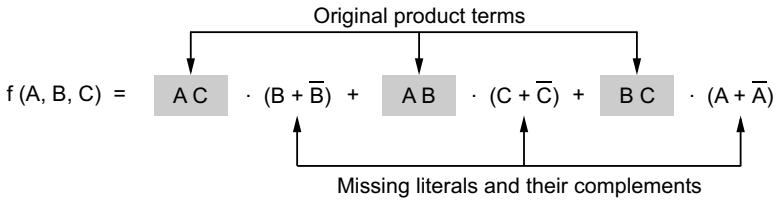
Example 1.4.1 *Convert the given expression in standard SOP form.*
f (A, B, C) = AC + AB + BC

Solution :

Step 1 : Find the missing literal/s in each product term.



Step 2 : AND product term with (missing literal + its complement).



Step 3 : Expand the terms and reorder literals.

Expand : f (A, B, C) = A C B + A C $\overline{B}$ + A B C + A B $\overline{C}$ + B C A + B C $\overline{A}$

Reorder : f (A, B, C) = A B C + A $\overline{B}$ C + A B C + A B $\overline{C}$ + A B C + $\overline{A}$ B C

Note : After having sufficient practice student should expand product term and reorder literals in it in a single step.

Step 4 : Omit repeated product terms.

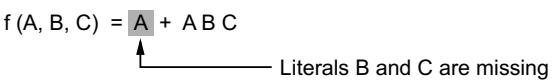
f (A, B, C) = A B C + A $\overline{B}$ C + A B C + A B $\overline{C}$ + A B C + $\overline{A}$ B C

$\therefore$ f (A, B, C) = A B C + A $\overline{B}$ C + A B $\overline{C}$ + $\overline{A}$ B C

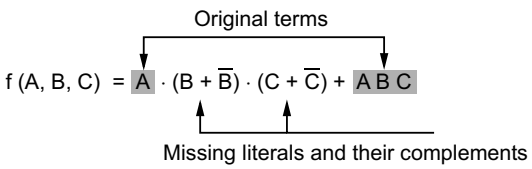
Example 1.4.2 *Convert the given expression in standard SOP form. f (A, B, C) = A + ABC*

Solution :

Step 1 : Find the missing literal/s in each product term.



Step 2 : AND product term with (missing literal + its complement).



Step 3 : Expand the terms and reorder literals.

f (A, B, C) = A B C + A B $\overline{C}$ + A $\overline{B}$ C + A $\overline{B}$ $\overline{C}$ + A B C

Step 4 : Omit repeated product term

Repeated product term

$$f(A, B, C) = A\,B\,C + A\,B\,\overline{C} + A\,\overline{B}\,C + A\,\overline{B}\,\overline{C} + A\,B\,C$$

$$\therefore\ f(A, B, C) = A\,B\,C + A\,B\,\overline{C} + A\,\overline{B}\,C + A\,\overline{B}\,\overline{C}$$

Example with Solution

Example 1.4.3 *Define canonical form. Express $F = BC' + AC$ in a canonical SOP form.*

Dec.-03, Marks 2

Solution :

$$\begin{aligned} F &= B\,\overline{C} + A\,C \\ &= \left(A + \overline{A}\right) B\,\overline{C} + A\,C\left(B + \overline{B}\right) \\ &= A\,B\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,C + A\,\overline{B}\,C \end{aligned}$$

Examples for Practice

Example 1.4.4 *Express the following function in standard SOP form*
$F_1 = AB + \overline{C}D + A\overline{B}C$

Ans. : $\overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}B\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,C\,D + A\,\overline{B}\,C\overline{D} + A\,\overline{B}\,C\,D + AB\,\overline{C}\,\overline{D} + AB\,\overline{C}D + ABC\overline{D} + ABCD$

Example 1.4.5 *Determine the canonical SOP form of* $f(x, y, z) = (x\,y + \overline{z})\,(y + x\,\overline{z})$

Ans. : $x\,y\,z + x\,y\,\overline{z} + \overline{x}\,y\,\overline{z} + x\,\overline{y}\,\overline{z}$

1.4.4.2 Steps to convert POS to standard POS form

Step 1 : Find the missing literals in each sum term if any.

Step 2 : OR each sum term having missing literal/s with term/s form by ANDing the literal and its complement.

Step 3 : Expand the terms by applying distributive law and reorder the literals in the sum terms.

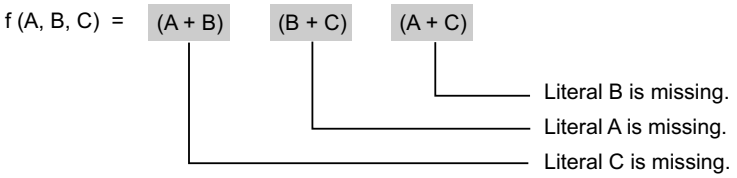
Step 4 : Reduce the expression by omitting repeated sum terms if any. Because $A \cdot A = A$.

Examples for Understanding

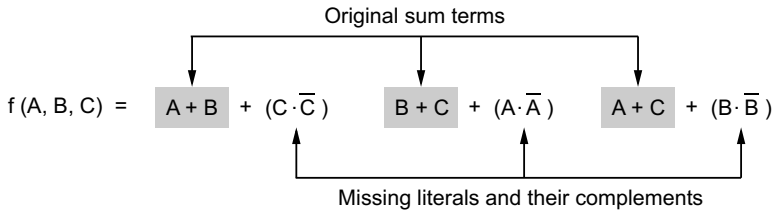
Example 1.4.6 *Convert the given expression in standard POS form.*
$f(A, B, C) = (A + B)(B + C)(A + C)$

Solution :

Step 1 : Find the missing literal/s in each sum term.



Step 2 : OR sum term with (missing literal • its complement).



Step 3 : Expand the terms and reorder literals.

Expand :

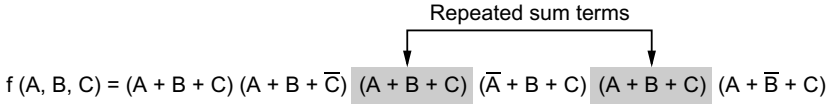
Since $A + BC = (A + B)(A + C)$ we have,

$$f(A, B, C) = (A + B + C)(A + B + \overline{C})(B + C + A)(B + C + \overline{A})$$
$$(A + C + B)(A + C + \overline{B})$$

Reorder :

$$f(A, B, C) = (A + B + C)(A + B + \overline{C})(A + B + C)(\overline{A} + B + C)$$
$$(A + B + C)(A + \overline{B} + C)$$

Step 4 : Omit repeated sum terms.



$\therefore$ $f(A, B, C) = (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)(A + \overline{B} + C)$

Example 1.4.7 *Convert the given expression in standard POS form.*

$Y = A \cdot (A + B + C)$

Solution :

Step 1 : Find the missing literal/s in each sum term.

$f(A, B, C) = A \cdot (A + B + C)$

Literals B and C are missing

Step 2 : OR sum term with (missing literal • its complement).

$$f(A, B, C) = (A + B \cdot \overline{B} + C \cdot \overline{C})(A + B + C)$$

Step 3 : Expand the terms and reorder literals.

Since $A + BC = (A + B)(A + C)$ we have,

$$f(A, B, C) = (A + B \cdot \overline{B} + C)(A + B \cdot \overline{B} + \overline{C})(A + B + C)$$

$$= (A + B + C)(A + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})(A + B + C)$$

Step 4 : Omit repeated sum terms.

Repeated sum term

$$f(A, B, C) = (A + B + C)(A + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})(A + B + C)$$

$$f(A, B, C) = (A + B + C)(A + \overline{B} + C)(A + B + \overline{C})(A + \overline{B} + \overline{C})$$

Examples with Solution

Example 1.4.8 *Convert the given expression in standard POS form.*

f(A, B, C) = (A + B) · (B + C)

Solution :

$$f(A, B, C) = [(A + B) + (C \cdot \overline{C})] \cdot [(B + C) + (A \cdot \overline{A})]$$

$$(A + B + C)(A + B + \overline{C})(A + B + C)(\overline{A} + B + C)$$

$$= (A + B + C)(A + B + \overline{C})(\overline{A} + B + C)$$

Example 1.4.9 *Convert SOP to equivalent POS.*

A'B'C + A'B'C +A'BC +AB'C +ABC **Dec.-07, Marks 4**

Solution : SOP $\rightarrow \overline{A}\,\overline{B}C + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$

$$\rightarrow \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

$$\text{POS} \rightarrow \left(\overline{A}+\overline{B}+\overline{C}\right)\left(\overline{A}+B+\overline{C}\right)\left(A+\overline{B}+\overline{C}\right)\left(A+B+\overline{C}\right)$$

Example 1.4.10 *Convert (A + B) (A + C) (B + $\overline{C}$) into standard POS form.*

May-08, Marks 7

Solution : Step 1 : OR sum term of missing literal

$$f(A, B, C) = [(A+B)+(C\cdot\overline{C})]\,[(A+C)+(B\cdot\overline{B})]\,[(B+\overline{C})+(A\cdot\overline{A})]$$

Step 2 : Expand the terms and reorder literals.

$$f(A, B, C) = (A+B+C)\,(A+B+\overline{C})\,(A+B+C)\,(A+\overline{B}+C)\,(A+B+\overline{C})\,(\overline{A}+B+\overline{C})$$

Step 3 : Omit repeated sum terms

$$f(A, B, C) = (A+B+C)\,(A+B+\overline{C})\,(A+\overline{B}+C)\,(\overline{A}+B+\overline{C})$$

Example 1.4.11 *Obtain the canonical POS for F(A, B, C) = (A + B′) (B + C) (A + C′)*

Dec.-08, Marks 6

Solution :

$$\begin{aligned} F(A, B, C) &= [(A+\overline{B})+(C+\overline{C})]\,[(B+C)+(A\cdot\overline{A})][(A+\overline{C})+(B\cdot\overline{B})] \\ &= (A+\overline{B}+C)\,(A+\overline{B}+\overline{C})\,(A+B+C)\,(\overline{A}+B+C) \\ &\quad (A+B+\overline{C})(A+\overline{B}+C) \\ &= (A+\overline{B}+C)\,(A+\overline{B}+\overline{C})\,(A+B+C)\,(\overline{A}+B+C)(A+B+\overline{C}) \end{aligned}$$

Example for Practice

Example 1.4.12 *Convert the given expression in standard POS form*
f (P, Q, R) = (P + $\overline{Q}$) (P + R)

Ans. : (P + $\overline{Q}$ + R) (P + $\overline{Q}$ + $\overline{R}$) (P + Q + R)

1.4.5 M Notations : Minterms and Maxterms

Each individual term in standard SOP form is called **minterm** and each individual term in standard POS form is called **maxterm**. The concept of minterms and maxterms allows us to introduce a very convenient shorthand notations to express logical functions. Table 1.4.1 gives the minterms and maxterms for a three literal/variable logical function where the number of minterms as well as maxterms is $2^3 = 8$. In general, for an n-variable logical function there are 2^n minterms and an equal number of maxterms.

Variables			Minterms	Maxterms
A	B	C	m_i	M_i
0	0	0	$\overline{A}\,\overline{B}\,\overline{C} = m_0$	$A + B + C = M_0$
0	0	1	$\overline{A}\,\overline{B}\,C = m_1$	$A + B + \overline{C} = M_1$
0	1	0	$\overline{A}\,B\,\overline{C} = m_2$	$A + \overline{B} + C = M_2$
0	1	1	$\overline{A}\,B\,C = m_3$	$A + \overline{B} + \overline{C} = M_3$
1	0	0	$A\,\overline{B}\,\overline{C} = m_4$	$\overline{A} + B + C = M_4$
1	0	1	$A\,\overline{B}\,C = m_5$	$\overline{A} + B + \overline{C} = M_5$
1	1	0	$A\,B\,\overline{C} = m_6$	$\overline{A} + \overline{B} + C = M_6$
1	1	1	$A\,B\,C = m_7$	$\overline{A} + \overline{B} + \overline{C} = M_7$

Table 1.4.1 Minterms and maxterms for three variables

As shown in Table 1.4.1 each minterm is represented by m_i and each maxterm is represented by M_i, where the subscript i is the decimal number equivalent of the natural binary number. With these shorthand notations logical function can be represented as follows :

1. $f(A, B, C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,\overline{C}$

$= m_0 + m_1 + m_3 + m_6$

$= \sum m(0, 1, 3, 6)$

2. $f(A, B, C) = (A + B + \overline{C})\,(A + \overline{B} + \overline{C})\,(\overline{A} + \overline{B} + C)$

$= M_1 \bullet M_3 \bullet M_6$

$= \Pi\, M\,(1, 3, 6)$

where $\sum$ denotes **sum of product** while Π denotes **product of sum**.

We know that logical expression can be represented in the truth table form. It is possible to write logic expression in standard SOP or POS form corresponding to a given truth table. The logic expression corresponding to a given truth table can be written in a standard sum of products form by writing one product term for each input combination that produces an output of 1. These product terms are ORed together to create the standard sum of products. The product terms are expressed by writing complement of a variable when it appears as an input 0, and the variable itself when it appears as an input 1. Consider, for example, the truth Table 1.4.2.

The product corresponding to input combination 010 is $\overline{A}B\overline{C}$, the product corresponding to input combination 011 is $\overline{A}BC$ and product corresponding to input combination 110 is $AB\overline{C}$. Thus the standard sum of products form is

$$f(A, B, C) = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$
$$= m_2 + m_3 + m_6$$

The logic expression corresponding to a truth table can also be written in a standard product of sums form by writing one sum term for each output 0. The sum terms are expressed by writing complement of a variable when it appears as an input 1 and the variable itself when it appears as an input 0. Consider, for example, the truth Table 1.4.3.

The sum corresponding to input combinations 010 is $A + \overline{B} + C$, and the sum corresponding to input 101 is $\overline{A} + B + \overline{C}$. Thus, the standard product of sum form is

$$f(A, B, C) = (A + \overline{B} + C)(\overline{A} + B + \overline{C})$$
$$= M_2 \bullet M_5$$

A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	1	$\leftarrow \overline{A}B\overline{C}$
0	1	1	1	$\leftarrow \overline{A}BC$
1	0	0	0	
1	0	1	0	
1	1	0	1	$\leftarrow AB\overline{C}$
1	1	1	0	

Table 1.4.2

A	B	C	Y	
0	0	0	1	
0	0	1	1	
0	1	0	0	$\leftarrow A + \overline{B} + C$
0	1	1	1	
1	0	0	1	
1	0	1	0	$\leftarrow \overline{A} + B + \overline{C}$
1	1	0	1	
1	1	1	1	

Table 1.4.3

1.4.6 Complements of Standard Forms

The POS and SOP functions derived from the same truth table are logically equivalent. In terms of minterms and maxterms we can then write

$$f(A, B, C) = m_0 + m_1 + m_3 + m_4 + m_6 + m_7 = M_2 + M_5$$
$$\therefore \quad f(A, B, C) = \sum m(0, 1, 3, 4, 6, 7) = \pi M(2, 5)$$

From the above expressions we can easily notice that there is a complementary type of relationship between a function expressed in terms of maxterms. Using this complementary relationship we can find logical function in terms of maxterms if function in minterms is known or vice-versa. For example, for a four variables if

$$f(A, B, C, D) = \sum m(0, 2, 4, 6, 8, 10, 12, 14)$$

then $f(A, B, C, D) = \pi M(1, 3, 5, 7, 9, 11, 13, 15)$

Examples with Solution

Example 1.4.13 *Express the switching function $f_{(BA)} = A$ in terms of minterms.*

Dec.-03, Marks 2

Solution :

$$f_{(BA)} = A$$
$$= A\ (B + \overline{B})$$
$$= AB + A\overline{B}$$

Example 1.4.14 *Express F = A + B'C as sum of minterms.* **May-11, Dec.-07, Marks 6**

Solution :

$$A + \overline{B}C = A\left(B + \overline{B}\right)\left(C + \overline{C}\right) + \left(A + \overline{A}\right)\overline{B}C$$
$$= \left(AB + A\overline{B}\right)\left(C + \overline{C}\right) + \left(A\ \overline{B}\ C\right) + \overline{A}\ \overline{B}C$$
$$= A\ BC + A\ \overline{B}C + A\ B\overline{C} + A\ \overline{B}\ \overline{C} + A\ \overline{B}C + \overline{A}\ \overline{B}C$$
$$= A\ BC + A\ \overline{B}C + A\ B\overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}C$$
$$\therefore \quad F = \sum m\ (1, 4, 5, 6, 7)$$

Example 1.4.15 *Prove that the logical sum of all minterms of a Boolean function of 2 variables is 1.*

Dec.-09, Marks 2

Solution : For two variables A and B minterms are :

$\overline{A}\ \overline{B}$, $\overline{A}$ B, A $\overline{B}$ and A B. The logical sum of these minterms is,

$$F = \overline{A}\ \overline{B} + \overline{A}\ B + A\ \overline{B} + AB$$
$$= \overline{A}\ (\overline{B} + B) + A\ (\overline{B} + B)$$
$$= \overline{A} + A \qquad \because \overline{B} + B = 1$$
$$= 1 \qquad \ldots \text{(Proved)} \qquad \because \overline{A} + A = 1$$

Example 1.4.16 *Express the Boolean function $F = XY + \overline{X}Z$ in product of maxterm.*

Dec.-09, Marks 6

Solution :

$$F = XY + \overline{X}Z$$
$$= XY\ (Z + \overline{Z}) + \overline{X}Z\ (Y + \overline{Y}) = XYZ + XY\overline{Z} + \overline{X}YZ + \overline{X}\ \overline{Y}Z$$
$$\therefore \quad F = \sum m(7,\ 6,\ 3,\ 1)$$

$= \Pi M (0, 2, 4, 5)$

$= (X + Y + Z)(X + \overline{Y} + Z)(\overline{X} + Y + Z)(\overline{X} + Y + \overline{Z})$

Example 1.4.17 *Express the Boolean function as*
(1) POS form (2) SOP form
$D = (A' + B)(B' + C)$ **May-10, Marks 4**

Solution :

1. POS form : $D = (\overline{A} + B)(\overline{B} + C)$

2. SOP form : $D = \overline{A}\,\overline{B} + \overline{A}\,C + B\,\overline{B} + BC$

$= \overline{A}\,\overline{B} + \overline{A}\,C + BC \qquad \because B\,\overline{B} = 0$

Review Questions

1. *Define switching function.*
2. *Define literal, product term and sum term.*
3. *Explain sum of product form.*
4. *What do you mean by standard SOP and POS forms.*
5. *Explain how to convert SOP or POS expressions in their standard forms.*
6. *What do you mean by minterms and maxterms ?*
7. *Differentiate between min term and max term.* **May-12, Marks 4**

1.5 Karnaugh Map Minimization

In the previous section we have seen that for simplification of Boolean expressions by Boolean algebra we need better understanding of Boolean laws, rules and theorems. During the process of simplification we have to predict each successive step. For these reasons, we can never be absolutely certain that an expression simplified by Boolean algebra alone is the simplest possible expression. On the other hand, the map method gives us a systematic approach for simplifying a Boolean expression. The map method, first proposed by Veitch and modified by Karnaugh, hence it is known as the **Veitch diagram** or the **Karnaugh map**.

1.5.1 One-Variable, Two-Variable, Three-Variable and Four-Variable Maps

The basis of this method is a graphical chart known as Karnaugh map (K-map). It contains boxes called cells. Each of the cell represents one of the 2^n possible products that can be formed from n variables. Thus, a 2-variable map contains $2^2 = 4$ cells, a 3-variable map contains $2^3 = 8$ cells and so forth. Fig. 1.5.1 shows outlines of 1, 2, 3 and 4-variable maps.

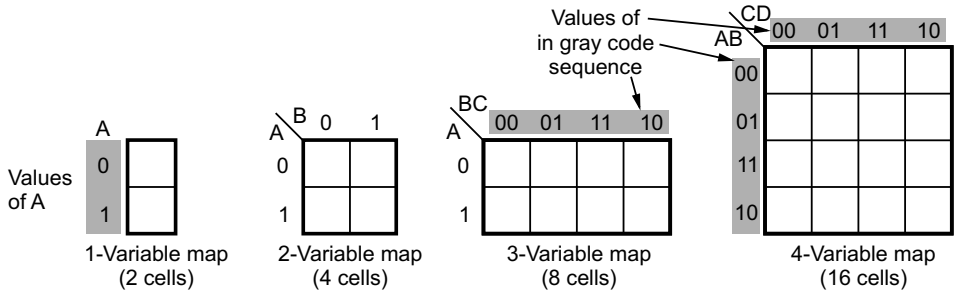


Fig. 1.5.1 Outlines of 1, 2, 3 and 4-variable Karnaugh maps

Product terms are assigned to the cells of a Karnaugh map by labeling each row and each column of the map with a variable, with its complement, or with a combination of variables and complements. The product term corresponding to a given cell is then the product of all variable in the row and column where the cell is located. Fig. 1.5.2 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps and the product terms corresponding to each cell.

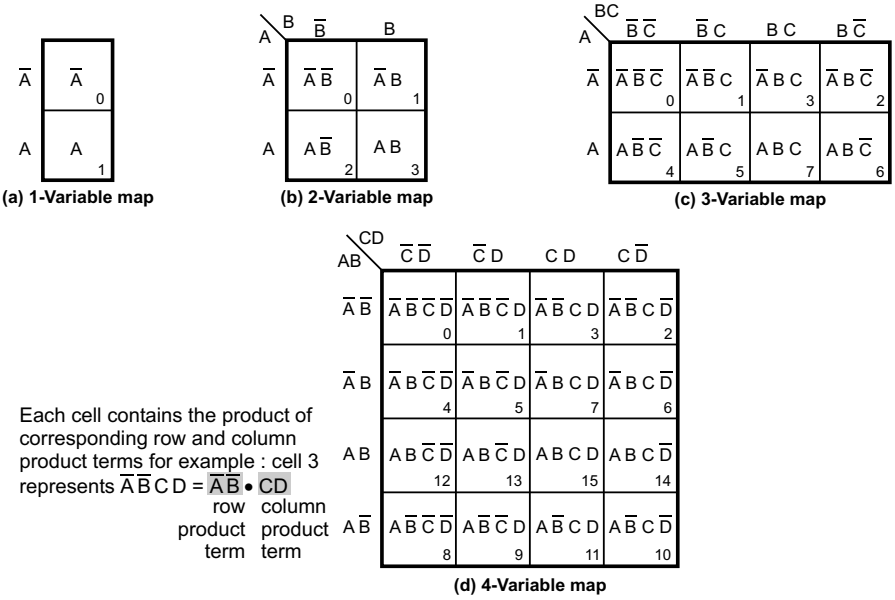

(a) 1-Variable map
(b) 2-Variable map
(c) 3-Variable map
(d) 4-Variable map
Each cell contains the product of corresponding row and column product terms for example : cell 3 represents $\overline{A}\,\overline{B}\,C\,D = \overline{A}\,\overline{B} \bullet CD$
row product term
column product term

Fig. 1.5.2 1, 2, 3 and 4-variable maps with product terms

It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complemented or to an uncomplemented form). For example, in Fig. 1.5.2 (b) the only change that occurs in moving along the bottom row from $A\overline{B}$ to AB is the change from $\overline{B}$ to B. Similarly, the only change that occurs in moving down the right column from $\overline{A}\,B$ to AB is the change from $\overline{A}$ to A. Irrespective of number of

variables the labels along each row and column must conform to the single-change rule. We know that the gray code has same properties (only one variable change when we proceed to next number or previous number) hence gray code is used to label the rows and columns of K-map as shown in Fig. 1.5.3.

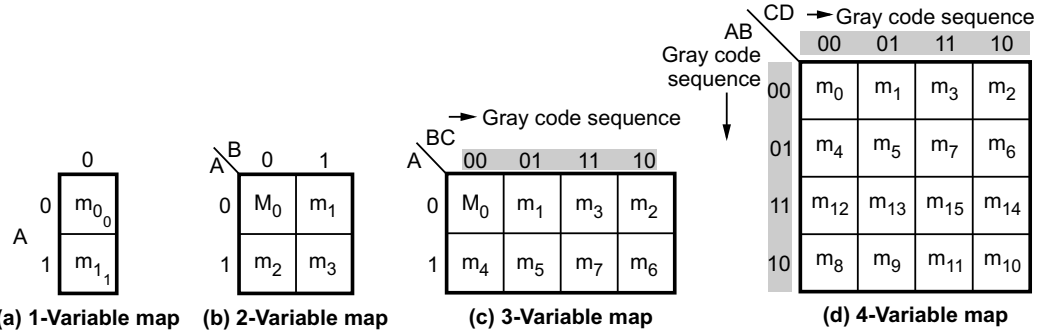


Fig. 1.5.3 Another way to represent 1, 2, 3 and 4-variable maps for SOP expressions

The Fig. 1.5.3 shows label of the rows and columns of a 1, 2, 3 and 4-variable maps using gray code and the product terms corresponding to each cell. Here, instead of writing actual product terms, corresponding shorthand minterm notations are written in the cell and row and columns are marked with gray code instead of variables.

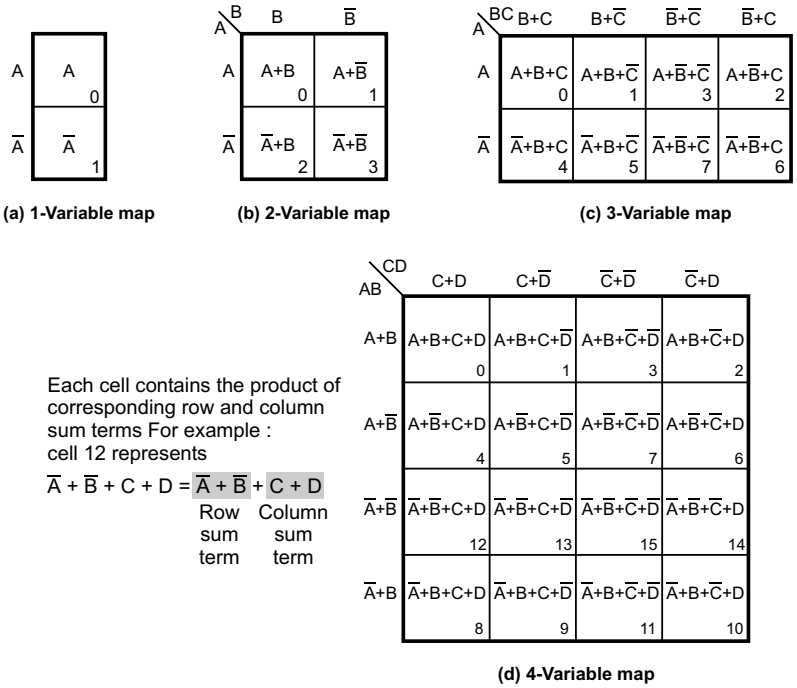

(a) 1-Variable map
(b) 2-Variable map
(c) 3-Variable map
Each cell contains the product of corresponding row and column sum terms For example :
cell 12 represents
Row sum term
Column sum term
(d) 4-Variable map

Fig. 1.5.4 1, 2, 3 and 4-variable maps with sum terms

In case of POS expressions we assign maxterms (sum terms) to the cells of a Karnaugh map. The Fig. 1.5.4 shows the way to label the rows and columns of a 1, 2, 3 and 4-variable maps and the sum terms corresponding to each cell. The Fig. 1.5.4 shows label of the rows and columns of 1, 2, 3 and 4-variable maps using gray code and the sum terms corresponding to each cell. Here, instead of writing actual sum terms, corresponding shorthand maxterm notations are written in the cell and row and columns are marked with gray code instead of variables.

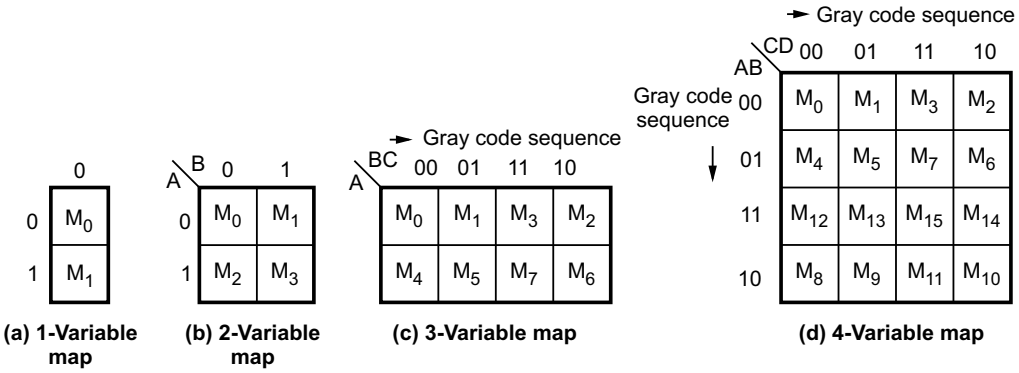


Fig. 1.5.5 Another way to represent 1, 2, 3 and 4-variable map for POS expressions

1.5.2 Plotting a Karnaugh Map

We know that logic function can be represented in various forms such as truth table, SOP Boolean expression and POS Boolean expression. In this section we will see the procedures to plot the given logic function in any form on the Karnaugh map.

1.5.2.1 Representation of Truth Table on Karnaugh Map

Cell : The smallest unit of a Karnaugh map, corresponding to one line of a truth table. The input variables are the cell's co-ordinates and the output variable is the cell's contents.

Fig. 1.5.6 shows K-maps plotted from truth tables with 2, 3 and 4-variables. Looking at the Fig. 1.5.6 we can easily notice that the terms which are having output 1, have the corresponding cells marked with 1s. The other cells are marked with zeros.

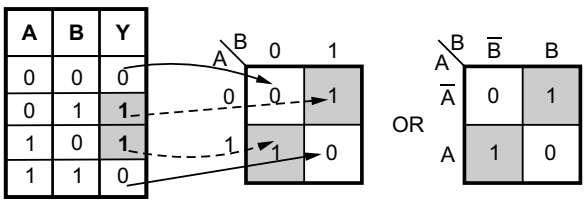
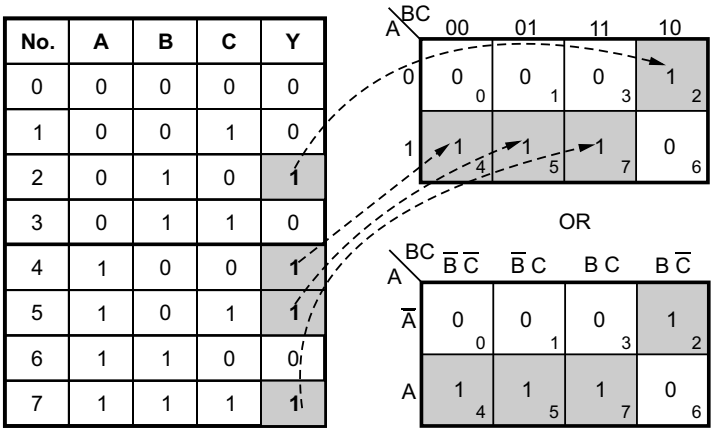


Fig. 1.5.6 (a) Representation of 2-variable truth table on K-map

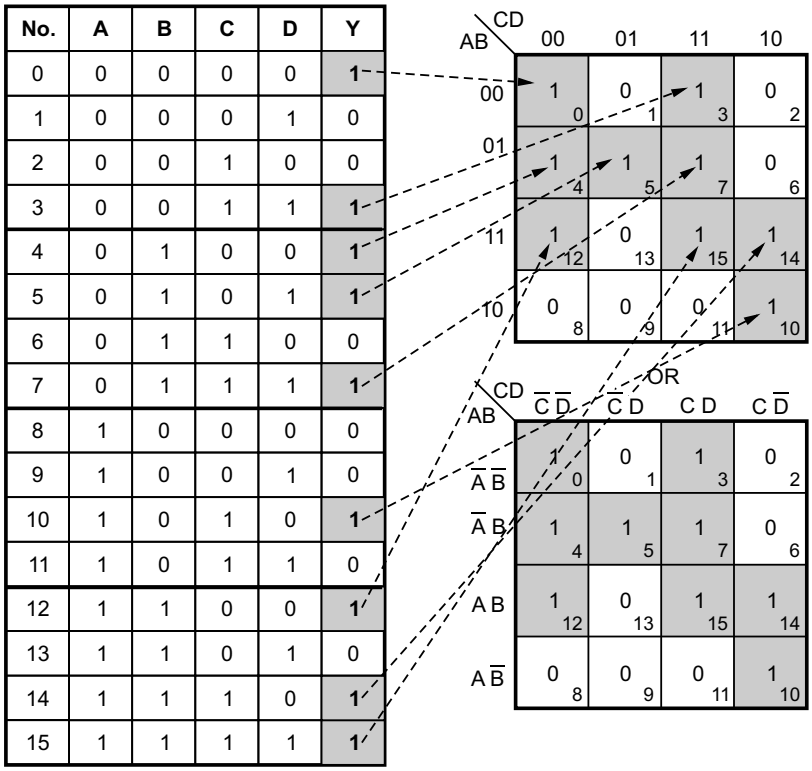
Note : The student can verify the data in each cell by checking the data in the column Y for particular row number and the data in the same cell number in the K-map.



No.	A	B	C	Y
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

A BC 00 01 11 10
0 0 0 0 1 2 0 1 3
1 1 1 1 0 4 5 7 6
OR
A BC $\overline{B}\,\overline{C}$ $\overline{B}$ C B C B $\overline{C}$
$\overline{A}$ 0 0 0 1 2 0 1 3
A 1 1 1 0 4 5 7 6

Fig. 1.5.6 (b) Representation of 3-variable truth table on K-map



No.	A	B	C	D	Y
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	0
14	1	1	1	0	1
15	1	1	1	1	1

(c) Representation of 4-variable truth table on K-map

Fig. 1.5.6 Plotting truth table on K-map

1.5.2.2 Representing Standard SOP on K-Map

A Boolean expression in the sum of products form can be plotted on the Karnaugh map by placing a 1 in each cell corresponding to a term (minterm) in the sum of products expression. Remaining cells are filled with zeros. This is illustrated in the following examples.

Examples for Understanding

Example 1.5.1 *Plot Boolean expression* $Y = AB\overline{C} + ABC + \overline{A}\,\overline{B}C$ *on the Karnaugh map.*

Solution : The expression has 3-variables and hence it can be plotted using 3-variable as in Fig. 1.5.7.

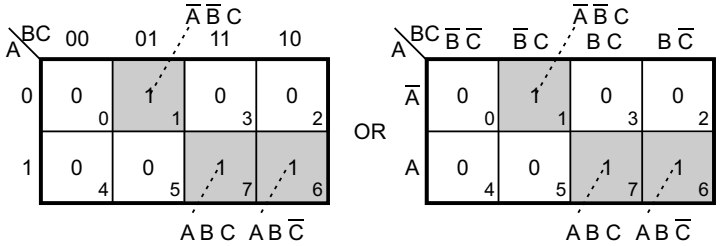


Fig. 1.5.7

Example 1.5.2 *Plot Boolean expression*
$Y = \overline{A}B\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}CD + AB\overline{C}D$ *on the Karnaugh map.*

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 1.5.8.

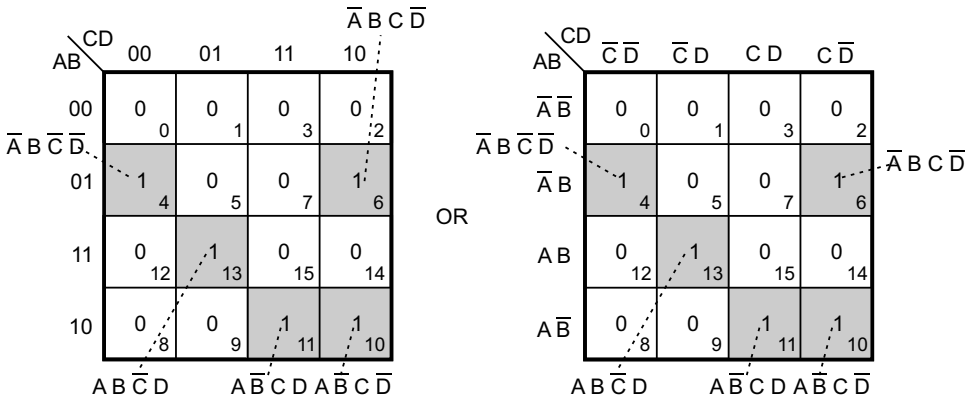


Fig. 1.5.8

1.5.2.3 Representing Standard POS on K-Map

A Boolean expression in the product of sums can be plotted on the Karnaugh map by placing a 0 in each cell corresponding to a term (maxterm) in the expression. Remaining cells are filled with ones. This is illustrated in the following examples.

Examples for Understanding

Example 1.5.3 *Plot Boolean expression* $Y = (A + \overline{B} + C)\ (A + \overline{B} + \overline{C})\ (\overline{A} + \overline{B} + C)\ (A + B + \overline{C})$ *on the Karnaugh map.*

Solution : The expression has 3-variables and hence it can be plotted using 3-variable map as shown in Fig. 1.5.9.

$$(A + \overline{B} + C) = M_2,\ (A + \overline{B} + \overline{C}) = M_3,\ (\overline{A} + \overline{B} + C) = M_6,\ (A + B + \overline{C}) = M_1$$

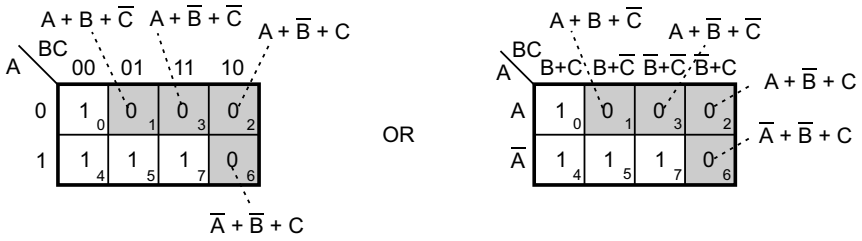


Fig. 1.5.9

Example 1.5.4 *Plot Boolean expression.*

$Y = (A + B + C + \overline{D})\ (A + \overline{B} + \overline{C} + D)\ (A + B + \overline{C} + \overline{D})\ (\overline{A} + \overline{B} + C + \overline{D})\ (\overline{A} + \overline{B} + \overline{C} + D)$

Solution : The expression has 4-variables and hence it can be plotted using 4-variable map as shown in Fig. 1.5.10.

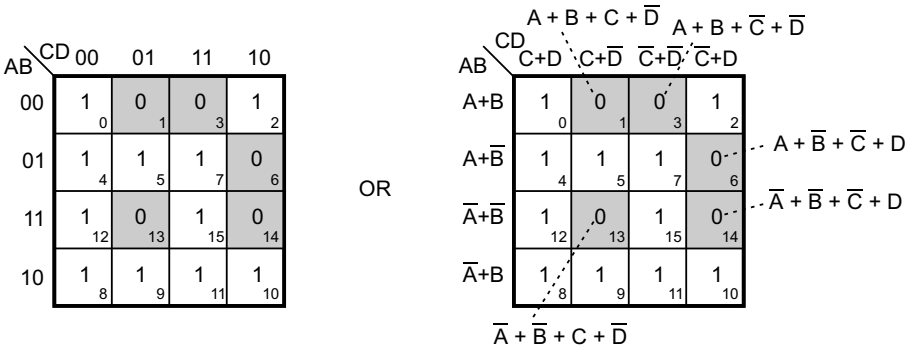


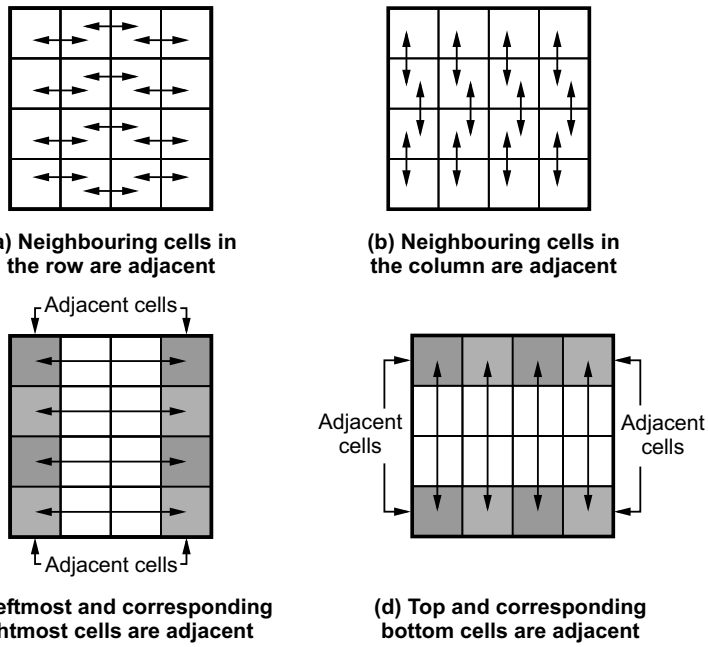
Fig. 1.5.10

$(A + B + C + \overline{D}) = M_1$, $(A + \overline{B} + \overline{C} + D) = M_6$, $(A + B + \overline{C} + \overline{D}) = M_3$,

$(\overline{A} + \overline{B} + C + \overline{D}) = M_{13}$, $(\overline{A} + \overline{B} + \overline{C} + D) = M_{14}$

1.5.3 Grouping Cells for Simplification

In the last section we have seen representation of Boolean function on the Karnaugh map. We have also seen that minterms are marked by 1s and maxterms are marked by 0s. Once the Boolean function is plotted on the Karnaugh map we have to use grouping technique to simplify the Boolean function. The grouping is nothing but combining terms in adjacent cells. Two cells are said to be adjacent if they conform the single change rule. i.e. there is only one variable difference between co-ordinates of two cells. For example, the cells for minterms ABC and $\overline{A}BC$ are adjacent. The Fig. 1.5.11 shows the adjacent cells. The simplification is achieved by grouping adjacent 1s or 0s in groups of 2^i, where i = 1, 2, … , n and n is the number of variables. When adjacent 1s are grouped then we get result in the sum of products form; otherwise we get result in the product of sums form. Let us see the various grouping rules.



(a) Neighbouring cells in the row are adjacent

(b) Neighbouring cells in the column are adjacent

(c) Leftmost and corresponding rightmost cells are adjacent

(d) Top and corresponding bottom cells are adjacent

Fig. 1.5.11 Adjacent cells

1.5.3.1 Grouping Two Adjacent Ones (Pair)

Fig. 1.5.12 (a) shows the Karnaugh map for a particular three variable truth table. This K-map contains a pair of 1s that are horizontally adjacent to each other; the first represents $\overline{A}\,\overline{B}\,C$ and the second represents $\overline{A}BC$. Note that in these two terms only the B variable appears in both normal and complemented form ($\overline{A}$ and C remain unchanged). Thus these two terms can be combined to give a resultant that eliminates

the B variable since it appears in both uncomplemented and complemented form. This is easily proved as follows :

$$Y = \overline{A}\,\overline{B}C + \overline{A}\,BC$$

$$= \overline{A}\,C\,(\overline{B}+B) \qquad \text{Rule 6 : } [A + \overline{A} = 1]$$

$$= \overline{A}\,C$$

This same principle holds true for any pair of vertically or horizontally adjacent 1s. Fig. 1.5.12 (b) shows an example of two vertically adjacent 1s. These two can be combined to eliminate A variable since it appears in both its uncomplemented and complemented forms. This gives result

$$Y = \overline{A}\,B\,C + A\,B\,C = B\,C$$

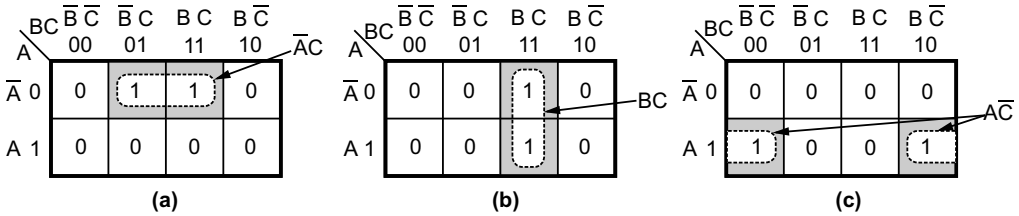


Fig. 1.5.12

In a Karnaugh map the corresponding cells in the leftmost column and rightmost column are considered to be adjacent. Thus, the two 1s in these columns with a common row can be combined to eliminate one variable. This is illustrated in Fig. 1.5.12 (c).

Here variable B has appeared in both its complemented and uncomplemented forms and hence eliminated as follows :

$$Y = A\,\overline{B}\,\overline{C} + A\,B\,\overline{C}$$

$$= A\,\overline{C}\,(\overline{B}+B) \qquad \text{Rule 6 : } [\overline{A} + A = 1]$$

$$= A\,\overline{C}$$

Let us see another example shown in Fig. 1.5.12 (d). Here two 1s from top row and bottom row of some column are combined to eliminate variable A, since in a K-map the top row and bottom row are considered to be adjacent.

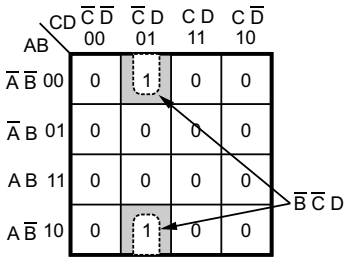


Fig. 1.5.12 (d)

$$Y = \overline{A}\,\overline{B}\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D$$

$$= \overline{B}\,\overline{C}\,D\,(\overline{A} + A)$$

$$= \overline{B}\,\overline{C}\,D$$

Fig. 1.5.12 (e) shows a Karnaugh map that has two overlapping pairs of 1s. This shows that we can share one term between two pairs.

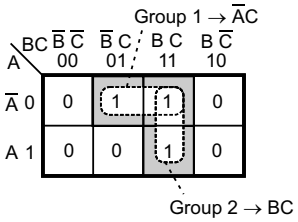


Fig. 1.5.12 (e)

$$Y = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C$$

$$= \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + \overline{A}\,B\,C + A\,B\,C \qquad \text{Rule 5 : } [A + A = A]$$

$$= \overline{A}\,C\,(\overline{B} + B) + B\,C\,(\overline{A} + A)$$

$$= \overline{A}\,C + B\,C$$

Fig. 1.5.12 (f) shows a K-map where three group of pairs can be formed. But only two pairs are enough to include all 1s present in the K-map. In such cases third pair is not required.

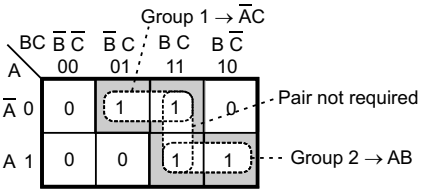

Group 1 → $\overline{A}C$
BC $\overline{B}\,\overline{C}$ $\overline{B}C$ BC $B\overline{C}$
00 01 11 10
A
$\overline{A}$ 0 0 1 1 0
A 1 0 0 1 1
Pair not required
Group 2 → AB

Fig. 1.5.12 (f) Examples of combining pairs of adjacent ones

$$Y = \overline{A}\,\overline{B}\,C + \overline{A}\,B\,C + A\,B\,C + A\,B\,\overline{C}$$

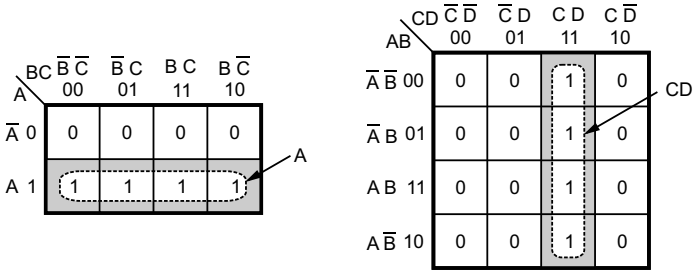
$$= \overline{A}\,C\,(\overline{B} + B) + A\,B\,(C + \overline{C}) \qquad \text{Rule 6 : } [\overline{A} + A = 1]$$

$$= \overline{A}\,C + A\,B$$

A pair is a group of two adjacent cells in a Karnaugh map. It cancels one variable in a K-map simplification.

1.5.3.2 Grouping Four Adjacent Ones (Quad)

In a Karnaugh map we can group four adjacent 1s. The resultant group is called Quad. Fig. 1.5.13 shows several examples of quads. Fig. 1.5.13 (a) shows the four 1s are horizontally adjacent and Fig. 1.5.13 (b) shows they are vertically adjacent.


BC $\overline{B}\,\overline{C}$ $\overline{B}C$ BC $B\overline{C}$
00 01 11 10
A
$\overline{A}$ 0 0 0 0 0
A 1 1 1 1 1
A
CD $\overline{C}\,\overline{D}$ $\overline{C}D$ CD $C\overline{D}$
00 01 11 10
AB
$\overline{A}\,\overline{B}$ 00 0 0 1 0
$\overline{A}B$ 01 0 0 1 0
AB 11 0 0 1 0
$A\overline{B}$ 10 0 0 1 0
CD

(a) Y = A **(b) Y = CD**

Fig. 1.5.13

A K-map in Fig. 1.5.13 (c) contains four 1s in a square and they are considered adjacent to each other. The four 1s in Fig. 1.5.13 (d) are also adjacent, as are those in Fig. 1.5.13 (e) because, as mentioned earlier, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

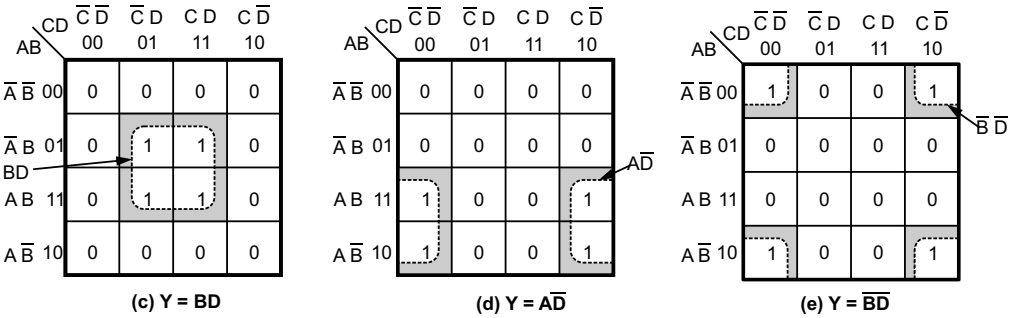


Fig. 1.5.13

From the above Karnaugh maps we can easily notice that when a quad is combined two variables are eliminated. For example, in Fig. 1.5.13 (c) we have following terms with 4 variables :

$$Y = \overline{A}\ B\ \overline{C}\ D + \overline{A}\ B\ C\ D + A\ B\ \overline{C}\ D + A\ B\ C\ D$$

$$= \overline{A}\ B\ D\ (\overline{C} + C) + A\ B\ D\ (\overline{C} + C)$$

$$= \overline{A}\ B\ D + A\ B\ D$$

$$= B\ D\ (\overline{A} + A)$$

$$= B\ D$$ (Only two variables in the result, variables A and C are eliminated)

Fig. 1.5.13 (f) shows overlapping groups. As mentioned earlier one term can be shared between two or more groups.

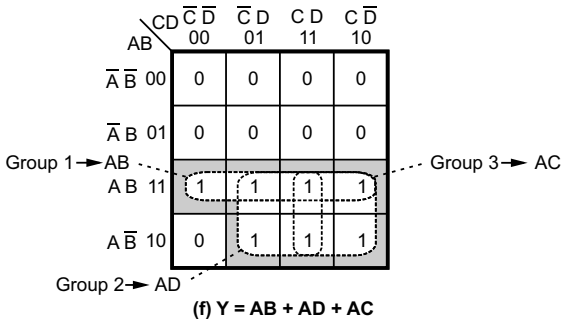


Fig. 1.5.13 Examples of combining quads of adjacent ones

Quad is a group of four adjacent cells in a Karnaugh map. It cancels two variables in a K-map simplification.

1.5.3.3 Grouping Eight Adjacent Ones (Octet)

In a Karnaugh map we can group eight adjacent 1s. The resultant group is called as octet. Fig. 1.5.14 shows several examples of octets. Fig. 1.5.14 (a) shows the eight 1s are horizontally adjacent and Fig. 1.5.14 (b) shows they are vertically adjacent.

From the Fig. 1.5.14 we can easily observe that when an octet is combined in a four variable map, three of the four variables are eliminated because only one variable remains unchanged. For example, in K-map shown in Fig. 1.5.14 (a) we have following terms :

$$
\begin{aligned}
Y &= \overline{A}\ B\ \overline{C}\ \overline{D} + \overline{A}\ B\ \overline{C}\ D + \overline{A}\ B\ C\ D + \overline{A}\ B\ C\ \overline{D} \\
&\quad + A\ B\ \overline{C}\ \overline{D} + A\ B\ \overline{C}\ D + A\ B\ C\ D + A\ B\ C\ \overline{D} \\
&= \overline{A}\ B\ \overline{C}\ (\overline{D} + D) + \overline{A}\ B\ C\ (D + \overline{D}) \\
&\quad + A\ B\ \overline{C}\ (\overline{D} + D) + A\ B\ C\ (D + \overline{D}) \\
&= \overline{A}\ B\ \overline{C} + \overline{A}\ B\ C + A\ B\ \overline{C} + A\ B\ C \quad = \overline{A}\ B\ (\overline{C} + C) + A\ B\ (\overline{C} + C) \\
&= \overline{A}\ B + A\ B \\
&= B\ (\overline{A} + A) = B
\end{aligned}
$$

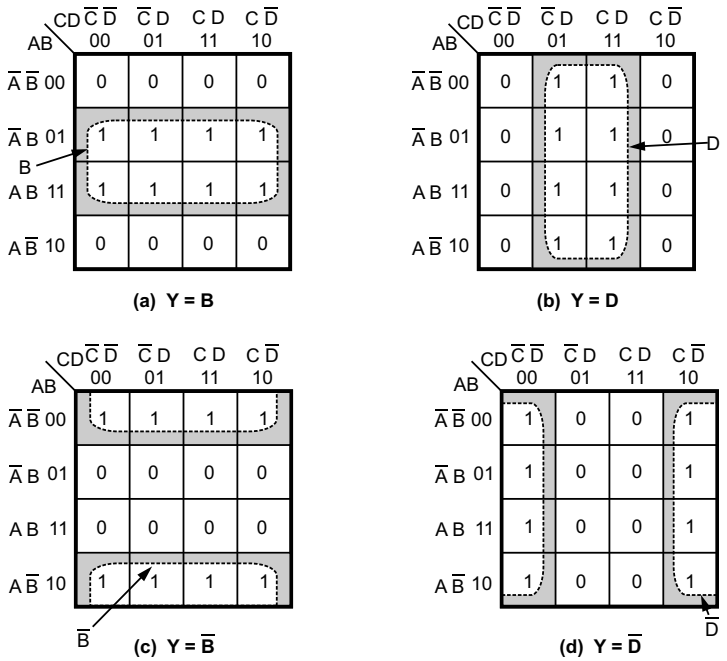
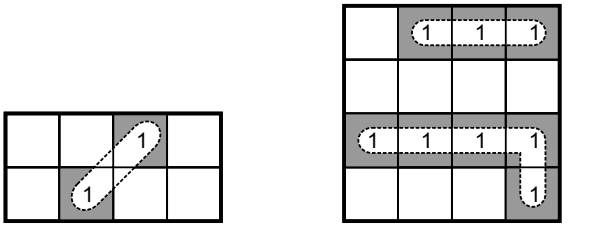


Fig. 1.5.14 Examples of combining octets of adjacent ones

Octet is a group of eight adjacent cells in a Karnaugh map. It cancels three variables in a K-map simplifications.

1.5.4 Illegal Grouping

The Fig. 1.5.15 shows the examples of illegal grouping of cells.



(a) Diagonal grouping is illegal

(b) Grouping of odd number of cells is illegal

Fig. 1.5.15

1.6 Simplification of SOP Expression

Dec.-03, 05, May-05, 06, 09, 10, 11

We have seen how combination of pairs, quads and octets on a Karnaugh map can be used to obtain a simplified expression. A pair of 1s eliminates one variable, a quad of 1s eliminates two variables and an octet of 1s eliminates three variables. In general, when a variable appears in both complemented and uncomplemented form within a group, that variable is eliminated from the resultant expression. Variables that are same in all with the group must appear in the final expression. Each group gives us a product term and summation of all product term gives us a Boolean expression. Therefore, we can say that, each product term implies the function and, hence is an implicant of the function. All the implicants of a function determined using a Karnaugh map are the prime implicants.

From the above discussion we can outline generalized procedure to simplify Boolean expressions as follows :

1. Plot the K-map and place 1s in those cells corresponding to the 1s in the truth table or sum of product expression. Place 0s in other cells.
2. Check the K-map for adjacent 1s and encircle those 1s which are not adjacent to any other 1s. These are called isolated 1s.
3. Check for those 1s which are adjacent to only one other 1 and encircle such pairs.
4. Check for quads and octets of adjacent 1s even if it contains some 1s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 1s that have not yet been grouped.
6. Form the simplified expression by summing product terms of all the groups.

Examples for Understanding

Example 1.6.1 *Minimize the expression* $Y = A\overline{B}\ C + \overline{A}\ \overline{B}\ C + \overline{A}BC + A\overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ \overline{C}$.

Solution :

Step 1 : Fig 1.6.1 (a) shows the K-map for three variables and it is plotted according to the given expression.

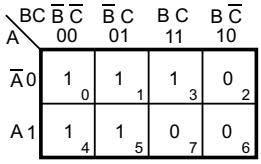


Fig. 1.6.1 (a)

Step 2 : There are no isolated 1s.

Step 3 : 1 in the cell 3 is adjacent only to 1 in the cell 1. This pair is combined and referred to as group 1.

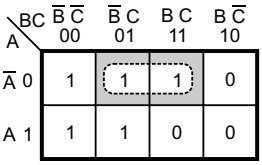


Fig. 1.6.1 (b)

Step 4 : There is no octet, but there is a quad. Cells 0, 1, 4 and 5 form a quad. This quad is combined and referred to as group 2.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 B variable is eliminated and in group 2 variables A and C are eliminated and we get,

$$Y = \overline{A}C + \overline{B}$$

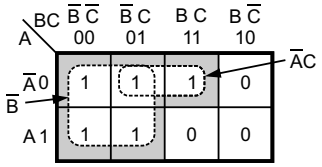


Fig. 1.6.1 (c)

Example 1.6.2 *Minimize the expression*

$$Y = \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + AB\overline{C}D + A\overline{B}\overline{C}D + \overline{A}\ \overline{B}\ C\overline{D}$$

Solution :

Step 1 : Fig. 1.6.2 (a) shows the K-map for four variables and it is plotted according to the given expression.

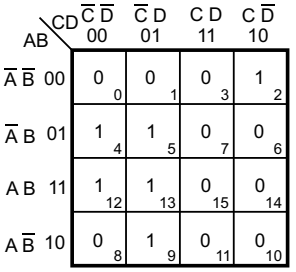


Fig. 1.6.2 (a)

Step 2 : Cell 2 is the only cell containing a 1 that is not adjacent to any other 1. It is referred to separately as group 1.

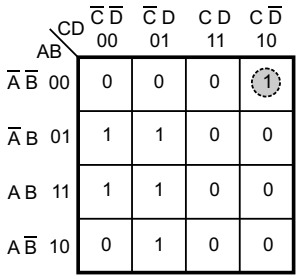


Fig. 1.6.2 (b)

Step 3 : 1 in the cell 9 is adjacent only to 1 in the cell 13. This pair is combined and referred to as group 2.

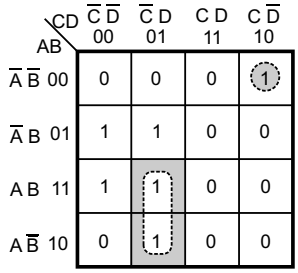


Fig. 1.6.2 (c)

Step 4 : There is no octet, but there is quad cells 4, 5, 12 and 13 form a quad. This quad is combined and referred to as group 3.

Step 5 : All 1s have already grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variable is not eliminated. In group 2 variable B is eliminated and in group 3 variables A and D are eliminated and we get,

$$Y = \overline{A}\,\overline{B}\,C\overline{D} + A\overline{C}\,D + B\overline{C}$$

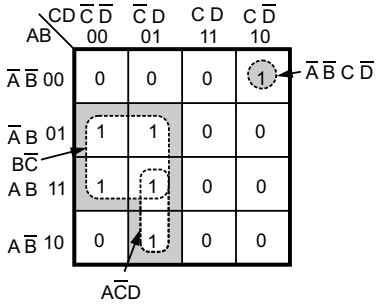


Fig. 1.6.2 (d)

Example 1.6.3 *Reduce the following four variable function to its minimum sum of products form :*

$$Y = \overline{A}\overline{B}C\overline{D} + ABC\overline{D} + A\overline{B}\,C\overline{D} + A\overline{B}\,CD + A\overline{B}\overline{C}\overline{D} + ABC\overline{D} + \overline{A}\overline{B}\,CD + \overline{A}\overline{B}C\overline{D}$$

Solution :

Step 1 : Fig. 1.6.3 (a) shows the K-map for four variables and it is plotted according to the given expression.

Step 2 : There are no isolated 1s.

Step 3 : There are no such 1s which are adjacent to only one other 1.

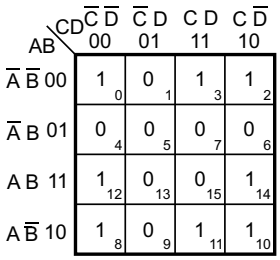

CD
AB
$\overline{C}\,\overline{D}$ 00
$\overline{C}$ D 01
C D 11
C $\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00
$\overline{A}$ B 01
A B 11
A $\overline{B}$ 10
1 0 0 1 1 3 1 2
0 4 0 5 0 7 0 6
1 12 0 13 0 15 1 14
1 8 0 9 1 11 1 10

Fig. 1.6.3 (a)

Step 4 : There are three quads formed by cells 0, 2, 8, 10, cells 8, 10, 12, 14 and cells 2, 3, 10, 11. These quads are combined and referred to as group 1, group 2 and group 3 respectively.

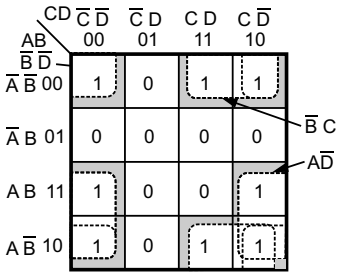


Fig. 1.6.3 (b)

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variables A and C are eliminated, in group 2 variables B and C are eliminated and in group 3 variables A and D are eliminated and we get,

$$Y = \overline{B}\,\overline{D} + A\overline{D} + \overline{B}C$$

Example 1.6.4 *Reduce the following function to its minimum sum of products form :*

$$Y = \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + \overline{A}BC\overline{D} + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD + A\overline{B}\,CD$$

Solution :

Step 1 : Fig. 1.6.4 (a) shows the K-map for four variables and it is plotted according to the given expression.

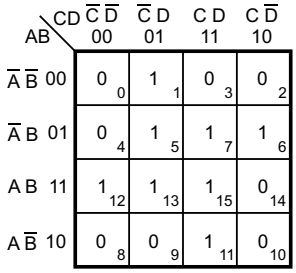


Fig. 1.6.4 (a)

Step 2 : There are no isolated 1s.

Step 3 : The 1 in the cell 1 is adjacent only to 1 in the cell 5, the 1 in the cell 6 is adjacent only to the 1 in the cell 7, the 1 in the cell 12 is adjacent only to the 1 in the cell 13 and the 1 in the cell 11 is adjacent only to the 1 in the cell 15. These pairs are combined and referred to as group 1-4 respectively.

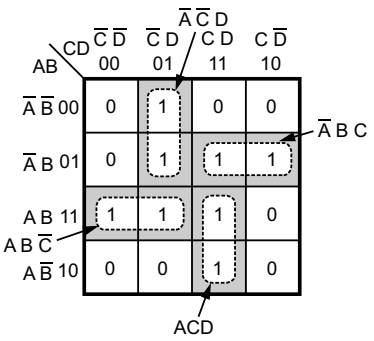


Fig. 1.6.4 (b)

Step 4 : There is no octet, but there is a quad. However, all 1s in the quad have already been grouped. Therefore this quad is ignored.

Step 5 : All 1s have already been grouped.

Step 6 : Each group generates a term in the expression for Y. In group 1 variable B is eliminated. Similarly, in group 2-4 variables D, D and B are eliminated one in each group. We finally get minimum sum of products form as,

$$Y = \overline{A}\,\overline{C}D + \overline{A}BC + AB\overline{C} + ACD$$

Example 1.6.5 *Simplify the logic function specified by the truth Table 1.6.1 using the Karnaugh map method. Y is the output variable, and A, B and C are the input variables.*

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 1.6.1

Solution :

Step 1 : Fig. 1.6.5 (a) shows the K-map for three variables and it is plotted according to given truth table.

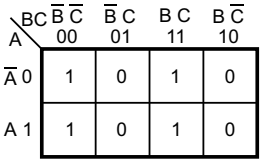


Fig. 1.6.5 (a)

Step 2 : There are no isolated 1s.

Step 3 : The 1 in the cell 0 is adjacent only to 1 in the cell 4 and the 1 in the cell 3 is adjacent only to 1 in the cell 7. These two pairs are grouped and referred to as group 1 and group 2.

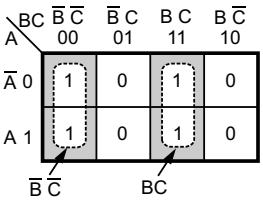


Fig. 1.6.5 (b)

Step 4 : There is no octet and quad.

Step 5 : All 1s have already been grouped.

Step 6 : In group 1 and group 2 variable A is eliminated and we get,

$$Y = \overline{B}\,\overline{C} + BC$$

Example 1.6.6 *Reduce the following function using Karnaugh map technique and implement using basic gates*

$f(A, B, C, D) = \overline{A}\overline{B}\,D + AB\overline{C}\,\overline{D} + \overline{A}BD + ABC\overline{D}$

Solution : The given function is not in the standard sum of products form. It is converted into standard SOP form as given below.

$$f(A, B, C, D) = \overline{A}\overline{B}D + AB\overline{C}\overline{D} + \overline{A}BD + ABC\overline{D}$$

$$= \overline{A}\overline{B}D(C + \overline{C}) + AB\overline{C}\overline{D} + \overline{A}BD(C + \overline{C}) + ABC\overline{D}$$

$$= \overline{A}\overline{B}CD + \overline{A}\overline{B}\overline{C}D + AB\overline{C}\overline{D} + \overline{A}BCD + \overline{A}BC\overline{D} + ABC\overline{D}$$

Step 1 : Fig. 1.6.6 (a) shows the K-map for four variables and it is plotted according to expression in standard SOP form.

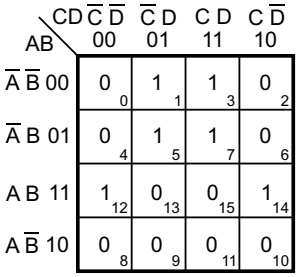


Fig. 1.6.6 (a)

Step 2 : There are no isolated 1s.

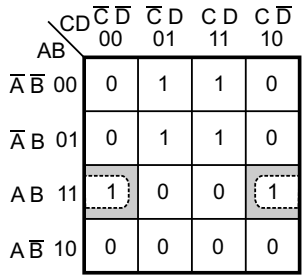


Fig. 1.6.6 (b)

Step 3 : The 1 in the cell 12 is adjacent only to the 1 in the cell 14. This pair is combined and referred to as group 1.

Step 4 : There is a quad. Cells 1, 3, 5 and 7 form a quad. This quad is referred to as group 2.

Step 5 : All 1s have already been grouped.

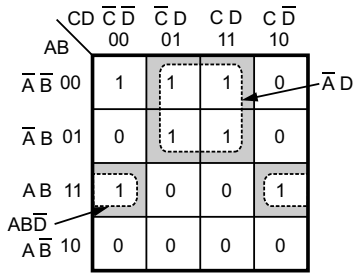


Fig. 1.6.6 (c)

Step 6 : In group 1 variable C is eliminated and in group 2 variables B and C are eliminated. We get simplified equation as,

$$Y = AB\overline{D} + \overline{A}D$$

Example 1.6.7 *Reduce the following function using K-map technique.*
$f(A, B, C, D) = \sum m\ (0, 1, 4, 8, 9, 10)$.

Solution :

Step 1 : Fig. 1.6.7 (a) shows the K-map for four variables and it is plotted according to given minterms.

Step 2 : There are no isolated 1s.

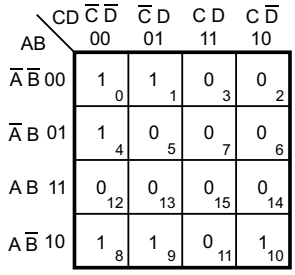


Fig. 1.6.7 (a)

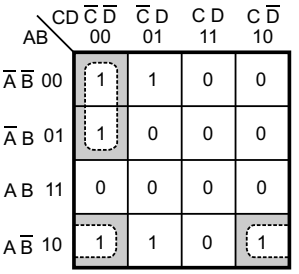

CD $\overline{C}\,\overline{D}$ $\overline{C}$ D C D C $\overline{D}$
AB 00 01 11 10
$\overline{A}\,\overline{B}$ 00 1 1 0 0
$\overline{A}$ B 01 1 0 0 0
A B 11 0 0 0 0
A $\overline{B}$ 10 1 1 0 1

Fig. 1.6.7 (b)

Step 3 : Cell 4 is adjacent only to cell 0 and cell 10 is adjacent only to cell 8. These two pairs are combined and referred to as group 1 and group 2 respectively.

Step 4 : There is a quad. Cells 0, 1, 8 and 9 form a quad. This quad is referred to as group 3.

Step 5 : All 1s have already been grouped.

Step 6 : In group 1, B and in group 2, C are eliminated respectively. In group 3, A and D are eliminated and finally we get

$$f(A, B, C, D) = \overline{A}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{D} + \overline{B}\,\overline{C}$$

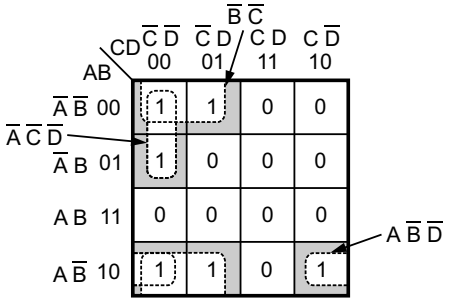
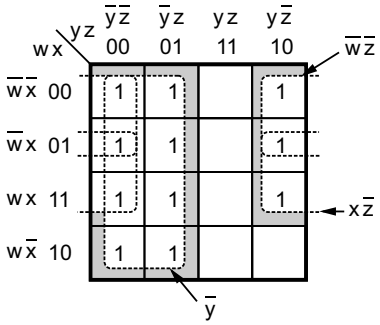

$\overline{B}\,\overline{C}$
CD $\overline{C}\,\overline{D}$ $\overline{C}$ D C D C $\overline{D}$
AB 00 01 11 10
$\overline{A}\,\overline{B}$ 00 1 1 0 0
$\overline{A}\,\overline{C}\,\overline{D}$
$\overline{A}$ B 01 1 0 0 0
A B 11 0 0 0 0
A $\overline{B}$ 10 1 1 0 1
A $\overline{B}$ $\overline{D}$

Fig. 1.6.7 (c)

Examples with Solutions

Example 1.6.8 *Plot the following Boolean function on a Karnaugh map and simplify it.*
$F(w, x, y, z) = \Sigma\,(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$ **Dec.-05, May-10, 11, Marks 6**

Solution :


yz $\overline{y}\,\overline{z}$ $\overline{y}$ z y z y $\overline{z}$
wx 00 01 11 10
$\overline{w}\,\overline{x}$ 00 1 1 1
$\overline{w}$ x 01 1 1 1
w x 11 1 1 1
w $\overline{x}$ 10 1 1
$\overline{w}\,\overline{z}$
x $\overline{z}$
$\overline{y}$

$\therefore f(w, x, y, z) = \overline{y} + x\,\overline{z} + \overline{w}\,\overline{z}$

Fig. 1.6.8

Example 1.6.9 *Show the Karnaugh map with the encircled groups for the Boolean function,* $F = \overline{C} + \overline{A}\,\overline{D} + A\,\overline{B}\,\overline{D}$. **May-06, Marks 2**

Solution :

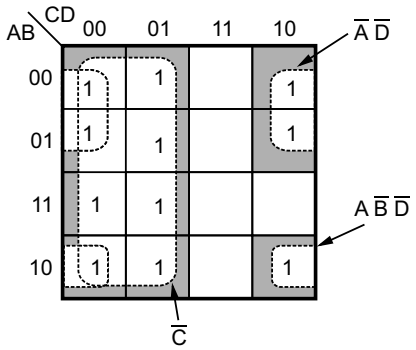


Fig. 1.6.9

Examples for Practice

Example 1.6.10 *Simplify following logical expression using Karnaugh maps*
$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,C + A\,B\,\overline{C}$ **Ans. : $Y = \overline{A}\,\overline{B} + \overline{C}$**

Example 1.6.11 *Simplify the following function*
$f_1\,(A, B, C, D) = \sum m\,(0, 3, 5, 6, 9, 10, 12, 15)$ **Ans. : $(A \oplus B) \odot (C \oplus D)$**

Example 1.6.12 *Simplify the following function*
$f_3\,(A, B, C, D) = \sum m\,(0, 1, 2, 3, 11, 12, 14, 15)$ **Ans. : $\overline{A}\,\overline{B} + A\,B\,\overline{D} + A\,C\,D$**

Example 1.6.13 *Simplify the following using K-map.*
$X = A'B + A'B'C + ABC' + AB'C'$ **Ans. : $X = \overline{A}C + B$**

Example 1.6.14 *Simplify the following function using K-map technique.*
$f(A, B, C, D) = \sum(0, 2, 3, 6, 7, 8, 10, 12, 13)$ **Ans. : $AB\overline{C} + \overline{B}\overline{D} + \overline{A}C$**

Example 1.6.15 *Simplify the following Boolean function using 4-variable map*
$f(w, x, y, z) = \Sigma\,(2, 3, 10, 11, 12, 13, 14, 15)$ **Ans. : $wx + \overline{x}y$**

1.6.1 Essential Prime Implicants

After grouping the cells, the sum terms which appear in the K-map are called prime implicants groups. It is observed that some cells may appear in only one prime implicants group; while other cells may appear in more than one prime implicants group. In Fig. 1.6.7 (c), cells 1, 4, 9 and 10 appear in only one prime implicants group. These cells are called **essential cells** and corresponding prime implicants are called **essential prime implicants**.

1.6.2 Incompletely Specified Functions (Don't Care Terms)

In some logic circuits, certain input conditions never occur, therefore the corresponding output never appears. In such cases the output level is not defined, it can be either HIGH or LOW. These output levels are indicated by `X' or `d' in the truth tables and are called **don't care outputs** or **don't care conditions** or **incompletely specified functions**. Let us see the output levels in the truth table as shown in the Table 1.6.2. Here outputs are defined for input conditions from 0 0 0 to 1 0 1. For remaining two conditions of input, output is not defined, hence these are called don't care conditions for this truth table.

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X

Table 1.6.2

A circuit designer is free to make the output for any "don't care" condition either a '0' or a '1' in order to produce the simplest output expression.

1.6.2.1 Describing Incomplete Boolean Function

We know that we describe the Boolean function using either a minterm canonical formula or a maxterm canonical formula. In order to obtain similar-type expressions for incomplete Boolean functions we use additional term to specify don't care conditions in the original expression. This is illustrated in the following examples.

In expression,

$$f(A, B, C) = \sum m (0, 2, 4) + d (1, 5)$$

minterms are 0, 2 and 4. The additional term d(1, 5) is introduced to specify the don't care conditions. This terms specifies that outputs for minterms 1 and 5 are not specified and hence these are don't care conditions. Letter d is used to indicate don't care conditions in the expression.

The above expression indicates how to represent don't care conditions in the minterm canonical formula. In the similar manner, we can specify the don't care conditions in the maxterm canonical formula. For example,

$$f(A, B, C) = \Pi M (2, 5, 7) + d(1, 3)$$

1.6.2.2 Don't Care Conditions in Logic Design

In this section, we see the example of incompletely specified Boolean function. Let us see the logic circuit for an even parity generator for 4-bit BCD number. The Table 1.6.3 shows the truth table for even-parity generator. The truth table shows that the output

for last six input conditions cannot be specified, because such input conditions does not occur when input is in the BCD form.

A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	-
1	0	1	1	-
1	1	0	0	-
1	1	0	1	-
1	1	1	0	-
1	1	1	1	-

Table 1.6.3 Truth table for even parity generator with 4-bit BCD input

The Boolean function for even parity generator with 4-bit BCD input can be expressed in minterm canonical formula as,

$$f(A, B, C, D) = \sum m (1, 2, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$$

1.6.2.3 Minimization of Incompletely Specified Functions

A circuit designer is free to make the output for any don't care condition either a '0' or '1' in order to produce the simplest output expression. Consider a truth table shown in Table 1.6.4. The K-map for this truth table is shown in Fig. 1.6.5 with x placed in the $AB\overline{C}$ and ABC cells.

It is not always advisable to put don't cares as 1s. This is illustrated in Fig. 1.6.10 (b). Here, the don't care output for cell ABC is taken as 1 to form a quad and don't care

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	X

Table 1.6.4

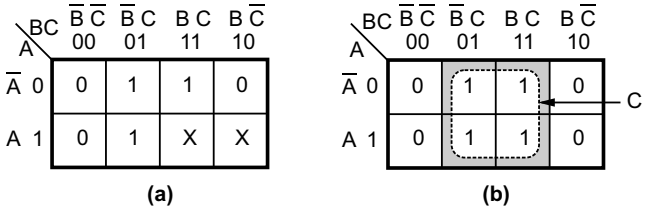


Fig. 1.6.10 Use of don't care conditions

output for cell $AB\overline{C}$ is taken as 0, since it is not helping any way to reduce an expression. Using don't care conditions in this way we get the simplified Boolean expression as

$$Y = C$$

From the above discussion we can realize that it is important to decide which don't cares to change to 0 and which to 1 to produce the best K-map grouping (i.e. the simplest expression). Now, we will see more examples to provide practice in dealing with "don't care" conditions.

Examples for Understanding

Example 1.6.16 *Find the reduced SOP form of the following function.*

$$f(A, B, C, D) = \sum m(1, 3, 7, 11, 15) + \sum d(0, 2, 4).$$

Solution :

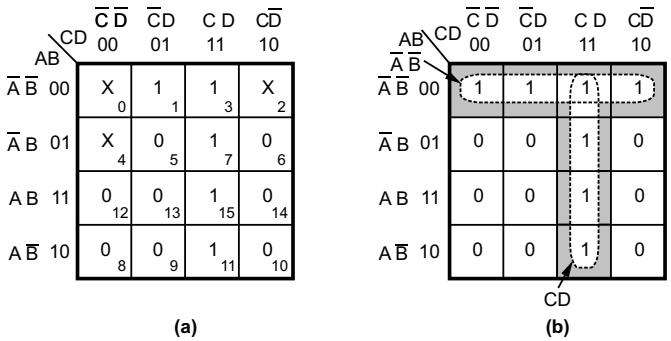


Fig. 1.6.11

To form a quad of cells 0, 1, 2 and 3 the don't care conditions 0 and 2 are replaced by 1s.

The remaining don't care condition is replaced by 0 since it is not required to form any group. With these replacements we get the simplified equation as

$$f(A, B, C, D) = \underbrace{\overline{A}\,\overline{B}}_{\text{Group 1}} + \underbrace{CD}_{\text{Group 2}}$$

Example 1.6.17 *Reduce the following function using Karnaugh map technique.*

$f(A, B, C, D) = \sum m(5, 6, 7, 12, 13) + \sum d(4, 9, 14, 15)$

May-09, Marks 8

Solution :

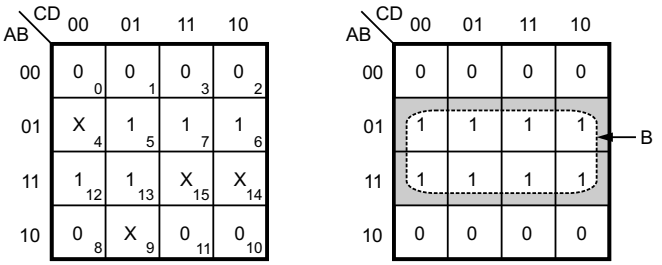


Fig. 1.6.12

To form a octet of cells 4, 5, 6, 7, 12, 13, 14 and 15 the don't care conditions 4, 14 and 15 are replaced by 1s. The remaining don't care condition 9 is replaced by 0 to get simplified function as,

$$f(A, B, C, D) = B$$

Example 1.6.18 *Reduce the following function using Karnaugh map technique.*

$f(A, B, C) = \sum m(0, 1, 3, 7) + \sum d(2, 5)$

Solution :

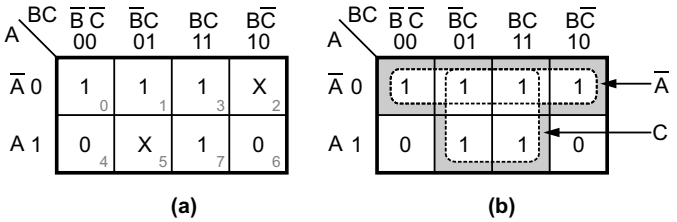


Fig. 1.6.13

To form two quads both the don't care conditions are replaced by 1s and we get,

$$f(A, B, C) = \overline{A} + C$$

Examples with Solutions

Example 1.6.19 *Reduce the following function using Karnaugh map technique.*
$F(W, X, Y, Z) = \sum m\ (0, 7, 8, 9, 10, 12) + \sum d\ (2, 5, 13).$

Solution :

In this example all don't cares are considered as 1s and we get,

$F(W, X, Y, Z) = \overline{X}\,\overline{Z} + \overline{W}\,X\,Z + W\overline{Y}$

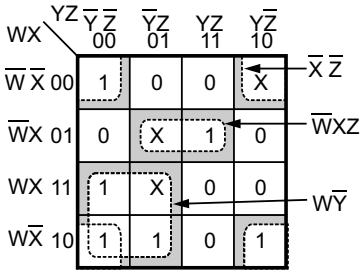


Fig. 1.6.14

Example 1.6.20 *Solve* $g(w, x, y, z) = \sum m(1, 3, 4, 6, 11) + \sum d(0, 8, 10, 12, 13)$

May-10, Marks 8

Solution :

$\therefore \quad g(w, x, y, z) = \overline{x}\,y\,z + \overline{w}\,x\,\overline{z} + \overline{w}\,\overline{x}\,z$

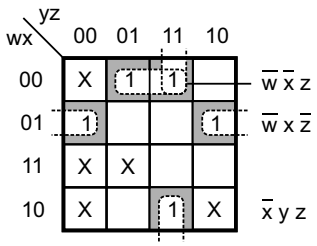


Fig. 1.6.15

Example 1.6.21 *Express the following function as the minimal sum of products using a K-map.* $f(a, b, c, d) = \Sigma\,(0,2,4,5,6,8,10,15) + \Sigma\phi(7,13,14)$ **May-05, Marks 12**

Solution :

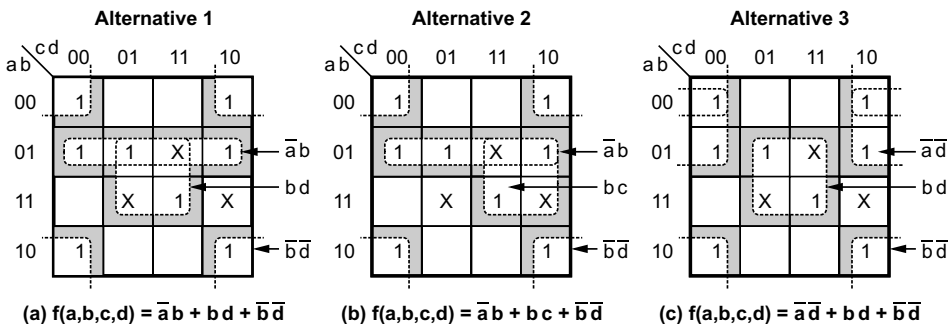


Fig. 1.6.16

As shown in Fig. 1.6.16 the given example has three solutions and all are correct. Students are expected to give any one solution.

Examples for Practice

Example 1.6.23 *Simplify the following switching function using Karnaugh map* $F(A,B,C,D) = \Sigma\ (0,5,7,8,9,10,11,14,15) + \phi\ (1,4,13)$.

Dec.-03, Marks 11

Ans. : $\overline{B}\ \overline{C}$ + BD + AC

Example 1.6.24 *Determine the minimal sum of product form of* $F(w,x,y,z) = \Sigma m\ (4,5,7,12,14,15) + d\ (3,8,10)$. **Ans. : $\overline{w}\ x\ \overline{y}$ + xyz + $w\overline{z}$**

Review Questions

1. *Give the steps for simplification of SOP expression.*
2. *What do you mean by essential prime implicants ?*
3. *What is don't care condition ?*

1.7 Simplification of POS Expressions

May-04, 09

In the above discussion, we have considered the Boolean expression in sum of products form and grouped 2, 4, and 8 adjacent ones to get the simplified Boolean expression in the same form. In practice, the designer should examine both the sum of products and product of sums reductions to ascertain which is more simplified. We have already seen the representation of product of sums on the Karnaugh map. Once the expression is plotted on the K-map instead of making the groups of ones, we have to make groups of zeros. Each group of zero results a sum term and it is nothing but the **prime implicate**. The technique for using maps for POS reductions is a simple step by step process and it is similar to the one used earlier.

1. Plot the K-map and place 0s in those cells corresponding to the 0s in the truth table or maxterms in the product of sums expression.
2. Check the K-map for adjacent 0s and encircle those 0s which are not adjacent to any other 0s. These are called isolated 0s.
3. Check for those 0s which are adjacent to only one other 0 and encircle such pairs.
4. Check for quads and octets of adjacent 0s even if it contains some 0s that have already been encircled. While doing this make sure that there are minimum number of groups.
5. Combine any pairs necessary to include any 0s that have not yet been grouped.
6. Form the simplified POS expression for F by taking product of sum terms of all the groups.

To get familiar with these steps we will solve some examples.

Examples for Understanding

Example 1.7.1 *Minimize the expression.*

$Y = (A + B + \overline{C}) (A + \overline{B} + \overline{C}) (\overline{A} + \overline{B} + \overline{C}) (\overline{A} + B + C) (A + B + C)$

Solution : $(A + B + \overline{C}) = M_1$, $(A + \overline{B} + \overline{C}) = M_3$, $(\overline{A} + \overline{B} + \overline{C}) = M_7$,

$(\overline{A} + B + C) = M_4$, $(A + B + C) = M_0$

Step 1 : Fig. 1.7.1 (a) shows the K-map for three variable and it is plotted according to given maxterms.

Step 2 : There are no isolated 0s.

Step 3 : 0 in the cell 4 is adjacent only to 0 in the cell 0 and 0 in the cell 7 is adjacent only to 0 in the cell 3. These two pairs are combined and referred to as group 1 and group 2 respectively.

Step 4 : There are no quads and octets.

Step 5 : The 0 in the cell 1 can be combined with 0 in the cell 3 to form a pair. This pair is referred to as group 3.

Step 6 : In group 1 and in group 2, A is eliminated, whereas in group 3 variable B is eliminated and we get,

$$Y = (B + C) (\overline{B} + \overline{C}) (A + \overline{C})$$

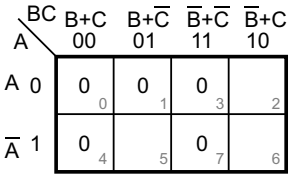


Fig. 1.7.1 (a)

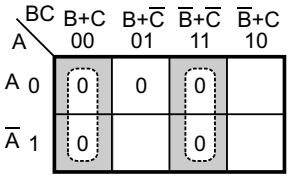


Fig. 1.7.1 (b)

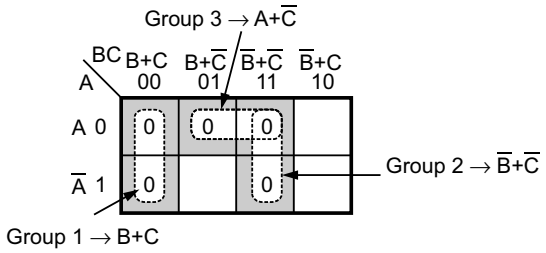


Fig. 1.7.1 (c)

Example 1.7.2 *Minimize the following expression in the POS form*

$Y = (\overline{A} + \overline{B} + C + D) (\overline{A} + \overline{B} + \overline{C} + D) (\overline{A} + \overline{B} + \overline{C} + \overline{D})$
$(\overline{A} + B + C + D) (A + \overline{B} + \overline{C} + D) (A + \overline{B} + \overline{C} + \overline{D}) (A + B + C + D)$
$(\overline{A} + \overline{B} + C + \overline{D})$

Solution :

$(\overline{A} + \overline{B} + C + D) = M_{12}$, $(\overline{A} + \overline{B} + \overline{C} + D) = M_{14}$, $(\overline{A} + \overline{B} + \overline{C} + \overline{D}) = M_{15}$

$(\overline{A} + B + C + D) = M_8$, $(A + \overline{B} + \overline{C} + D) = M_6$, $(A + \overline{B} + \overline{C} + \overline{D}) = M_7$

$(A + B + C + D) = M_0$ and $(\overline{A} + \overline{B} + C + \overline{D}) = M_{13}$

Step 1 : Fig. 1.7.2 (a) shows the K-map for four variable and it is plotted according to given maxterms.

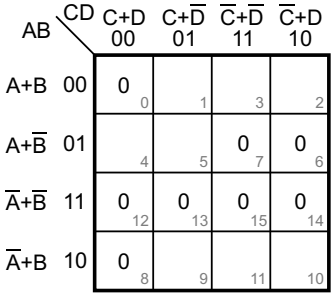


Fig. 1.7.2 (a)

Step 2 : There are no isolated 0s.

Step 3 : 0 in the cell 0 is adjacent only to 0 in the cell 8. This pair is combined and referred to as group 1.

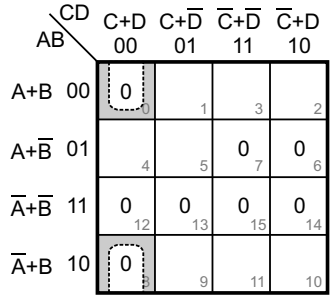


Fig. 1.7.2 (b)

Step 4 : There are two quads. Cells 12, 13, 14 and 15 forms a quad 1 and cells 6, 7, 14, 15 forms a quad 2. These two quads are referred to as group 2 and group 3, respectively.

Step 5 : All 0s have already been grouped.

Step 6 : In group 1, variable A is eliminated. In group 2, variable C and D are eliminated and in group 3 variables A and D are eliminated. Therefore we get simplified POS expression as,

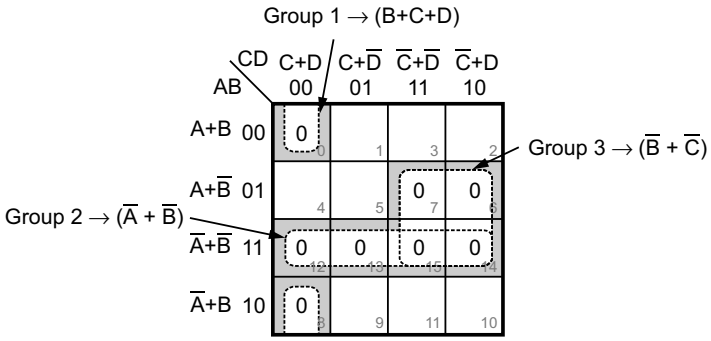


Fig. 1.7.2 (c)

$$Y = (B + C + D)(\overline{A} + \overline{B})(\overline{B} + \overline{C})$$

Example 1.7.3 *Reduce the following function using K-map technique*
$f(A, B, C, D) = \Pi M(0, 2, 3, 8, 9, 12, 13, 15)$

Solution :

Step 1 : Fig. 1.7.3 (a) shows the K-map for four variables and it is plotted according to given maxterms.

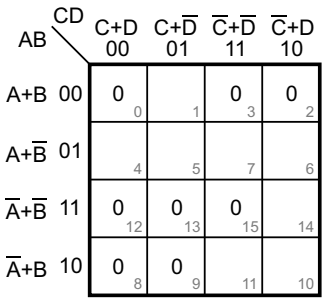


Fig. 1.7.3 (a)

Step 2 : There are no isolated 0s.

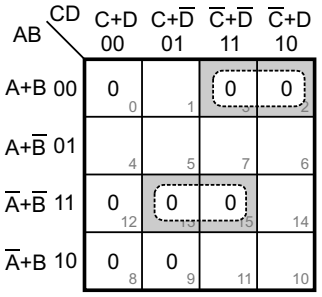


Fig. 1.7.3 (b)

Step 3 : The 0 in the cell 15 is adjacent only to 0 in the cell 13 and 0 in the cell 3 is adjacent only to 0 in the cell 2. These two pairs are combined and referred to as group 1 and group 2, respectively.

Step 4 : The cells 8, 9, 12 and 13 form a quad which is referred to as group 3.

Step 5 : The remaining 0 in the cell 0 is combined with the 0 in the cell 2 to form a pair, which is referred to as group 4.

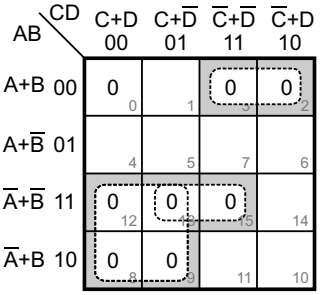


Fig. 1.7.3 (c)

Step 6 : In group 1 and in group 4 variable C is eliminated. In group 2 variable D is eliminated and in group 3 variables B and D are eliminated. Therefore, we get simplified expression in POS form as,

$$f = (\overline{A} + \overline{B} + \overline{D})\,(A + B + \overline{C})\,(\overline{A} + C)\,(A + B + D)$$

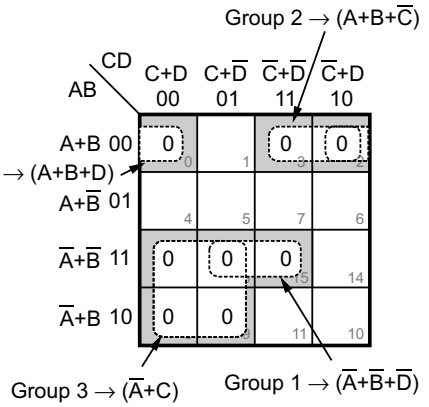


Fig. 1.7.3 (d)

Examples with Solutions

Example 1.7.4 *Simplify using K-map to obtain a minimum POS expression :*

$(\overline{A}+\overline{B}+C+D)(A+\overline{B}+C+D)(A+B+C+\overline{D})(A+B+\overline{C}+\overline{D})(\overline{A}+B+C+\overline{D})(A+B+\overline{C}+D)$ **May-04**

Solution : $Y = (\overline{B}+C+D)(B+C+\overline{D})(A+B+\overline{C})$

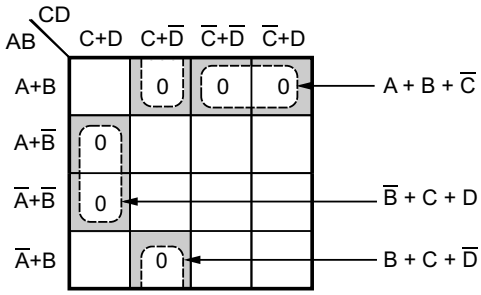


Fig. 1.7.4

Example 1.7.5 *Obtain the minimal product of sums for* $F = \sum (0,2,3,6,7) + d(8,10,11,15)$

Solution :

$$F = \sum (0, 2, 3, 6, 7) + d\ (8, 10, 11, 15)$$

$$= \pi\ (1, 4, 5, 9, 12, 13, 14) + d\ (8, 10, 11, 15)$$

$$\therefore \quad F = (C+\overline{D}) \cdot (\overline{B}+C) \cdot \overline{A}$$

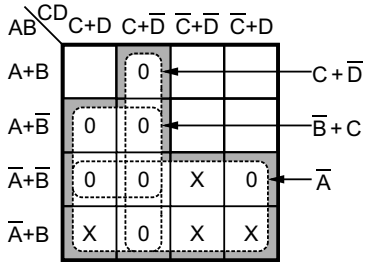


Fig. 1.7.5

Example 1.7.6 *Reduce the following function using K-map technique.*

f (A, B, C, D) = π (0, 3, 4, 7, 8, 10, 12, 14) + d (2, 6) **Dec.-09, Marks 10**

Solution :

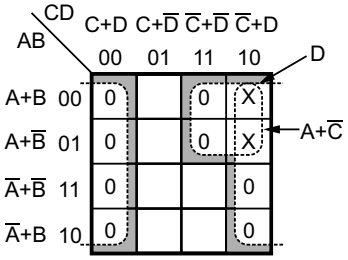


Fig. 1.7.6

$\therefore$ f(A, B, C, D) = D · (A + $\overline{C}$)

Examples for Practice

Example 1.7.7 *Simplify the following Boolean function for minimal POS form*
$F(w, x, y, z) = \pi M (4, 5, 6, 7, 8, 12) + d (1, 2, 3, 9, 11, 14)$

Ans. : $(w + \overline{x})\ (\overline{w} + y + z)$

Example 1.7.8 *Reduce the following function using K-map*
$F (A, B, C) = \pi M (0, 1, 2, 3, 4, 7)$ **Ans. : $F = A(\overline{B} + \overline{C}) \cdot (B + C)$**

Example 1.7.9 *Simplify the following expression using Boolean algebra.*
$f(x, y, z) = \pi M (3, 5, 7)$ **May-09, Marks 8**

Ans. : $(\overline{y} + \overline{z})\ (\overline{x} + \overline{z})$

Review Question

1. *Give the steps for simplification of POS expression.*

1.8 Five-Variable K-Map

Dec.-03, 11, May-08

A 5-variable K-map requires $2^5 = 32$ cells, but adjacent cells are difficult to identify on a single 32-cell map. Therefore, two 16-cell K-maps are generally used. If the variables are A, B, C, D and E two identical 16-cell maps containing B, C, D and E can be constructed. One map is then used for A and the other for $\overline{A}$. See Fig. 1.8.1. It is important to note that in order to identify the adjacent grouping in the five variable map, we must imagine the two maps superimposed on one another; not "hinged" or "mirror imaged". Every cell in one map is adjacent to the corresponding cell in the other map, because only one variable (i.e. A) changes between such corresponding cells. Thus, every row on one map is adjacent to the corresponding row (the one occupying the same position) on the other map, as are corresponding columns. Also, the rightmost and leftmost columns within each 16-cell map are adjacent, just as they are in any 16-cell map, as are the top and bottom rows. However, the rightmost column of one map is not adjacent to the leftmost column of the other map, since those are not corresponding columns. Nor is the top row of one map adjacent to the bottom row of the other. This is illustrated in Fig. 1.8.2.

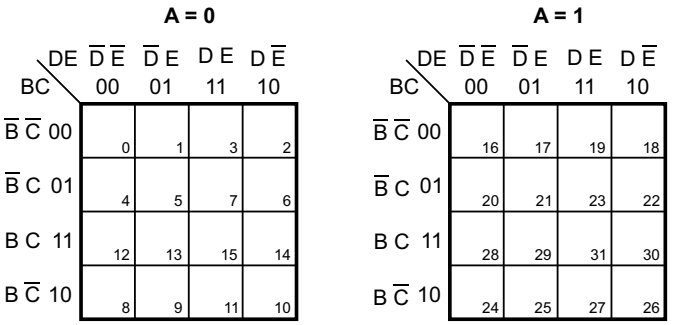

A = 0
DE $\overline{D}\overline{E}$ $\overline{D}$E DE D$\overline{E}$
BC 00 01 11 10
$\overline{B}\overline{C}$ 00 0 1 3 2
$\overline{B}$C 01 4 5 7 6
BC 11 12 13 15 14
B$\overline{C}$ 10 8 9 11 10
A = 1
DE $\overline{D}\overline{E}$ $\overline{D}$E DE D$\overline{E}$
BC 00 01 11 10
$\overline{B}\overline{C}$ 00 16 17 19 18
$\overline{B}$C 01 20 21 23 22
BC 11 28 29 31 30
B$\overline{C}$ 10 24 25 27 26

Fig. 1.8.1 Blank five-variable Karnaugh map

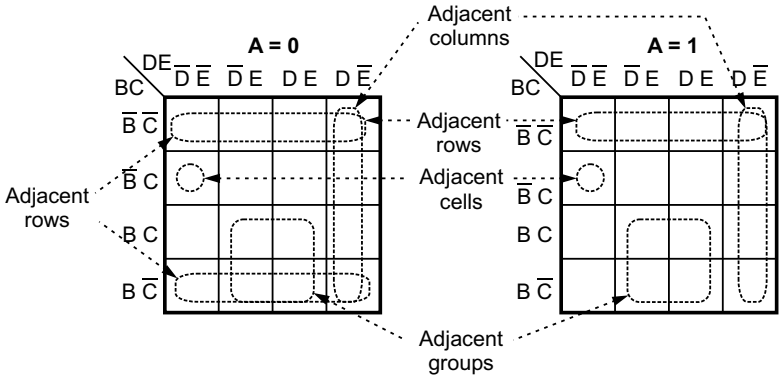


Fig. 1.8.2 A 5-variable Karnaugh map with examples of adjacencies

Examples for Understanding

Example 1.8.1 *Simplify the Boolean function.*

$f(A, B, C, D, E) = \sum m\,(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$

Solution :

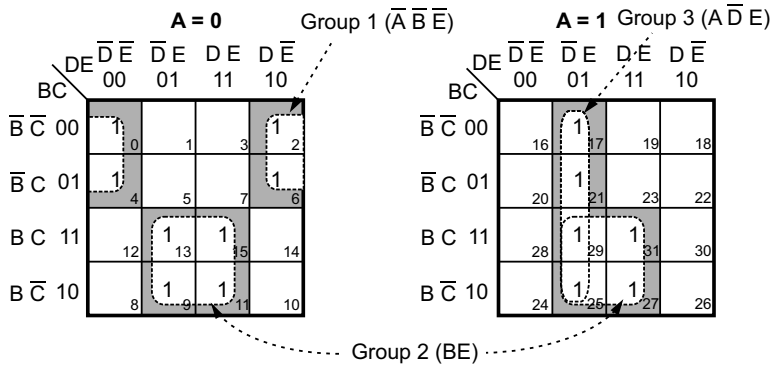


Fig. 1.8.3

The first 16-cell map has 2 quads. The second quad is adjacent to quad in the second 16-cell map. These two quads are grouped and referred to as group 2. The second 16-cell map has another quad which is referred to as group 3. Therefore, the simplified expression is

$$f(A, B, C, D, E) = \overline{A}\,\overline{B}\,\overline{E} + BE + A\overline{D}\,E$$

Example 1.8.2 *Simplify the Boolean function.*

$F(A, B, C, D, E) = \sum m\,(0, 5, 6, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27, 29, 31)$

Solution : As shown in the Fig. 1.8.4 there are two isolated 1s. These 1s are referred as group 2 and group 3, respectively. Group 1 is formed by combining two adjacent pair of 1s from two different 16-cell maps. The combination of two quads from two different

16-cell maps gives the group 4. The second 16-cell map has one pair and another quad which form group 5 and group 6 respectively. Therefore the simplified expression is

$$F(A, B, C, D, E) = \overline{C}\,\overline{D}\,\overline{E} + \overline{A}\,\overline{B}\,C\,\overline{D}\,E + \overline{A}\,\overline{B}\,C\,D\,\overline{E} + B\,\overline{C} + A\,\overline{B}\,\overline{D}\,\overline{E} + ABE$$

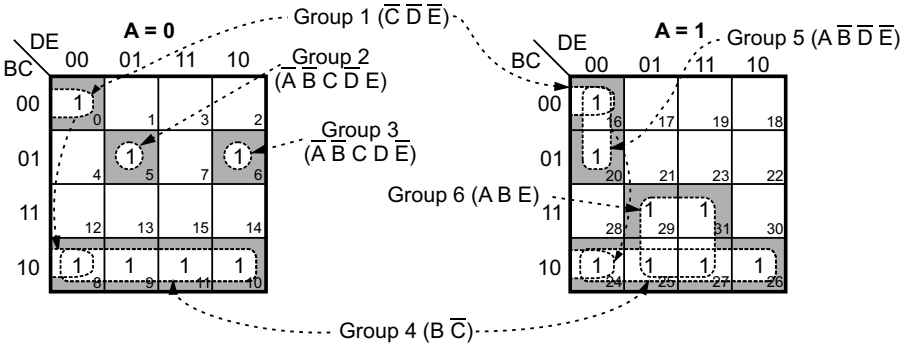


Fig. 1.8.4

Example 1.8.3 *Reduce the function using K-map technique*

$$F(A, B, C, D, E) = \sum m\,(1, 4, 8, 10, 11, 20, 22, 24, 25, 26) + d\,(0, 12, 16, 17)$$

Solution : The given function is plotted on the K-map with don't care conditions as shown in the Fig. 1.8.5. Here don't care conditions are taken as 1s to form pairs and quads. The group 1 is formed by combination of two pairs from two different 16-cell maps. Group 2 is quad within the first 16-cell map group 3 is formed by another combination of two pairs from two different 16-cell maps. The first 16-cell has one more pair of 1s which is referred to as group 4. Group 5 and group 6 are the quad and pair from the second 16-cell map, respectively.

Therefore, we get simplified expression as

$$F(A, B, C, D, E) = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{D}\,\overline{E} + B\,\overline{C}\,\overline{E} + \overline{A}\,B\,\overline{C}\,D + A\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{E}$$

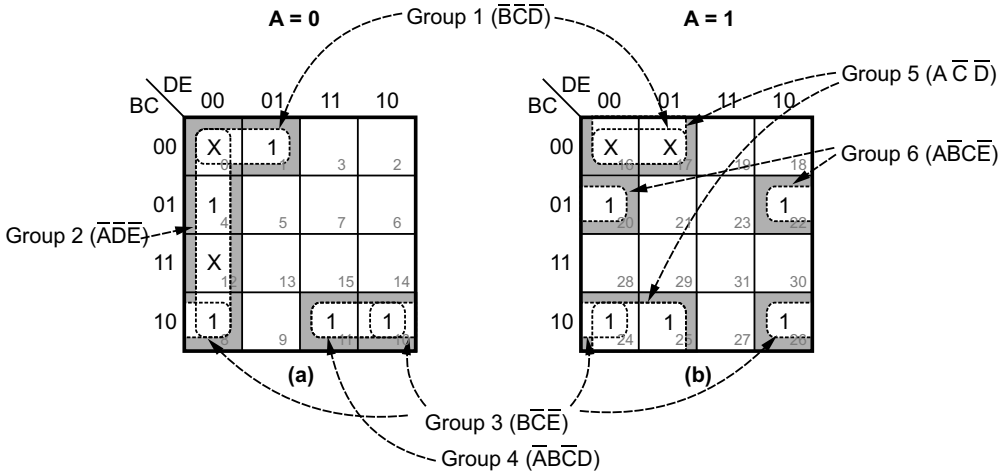


Fig. 1.8.5

Examples with Solutions

Example 1.8.4 *Simplify the following switching function*

$f(x_1, x_2, x_3, x_4, x_5) = \Sigma m(1, 3, 6, 10, 11, 12, 14, 15, 17, 19, 20, 22, 24, 29, 30)$

May-08, Marks 8

Solution :

$$f(x_1, x_2, x_3, x_4, x_5) = \bar{x}_2\bar{x}_3x_5 + x_1\bar{x}_2x_3\bar{x}_5 + x_1x_3x_4\bar{x}_5 + x_1x_2x_3\bar{x}_4x_5 + x_1x_2\bar{x}_3\bar{x}_4\bar{x}_5 + \bar{x}_1x_2x_3\bar{x}_5 + \bar{x}_1x_2x_4$$

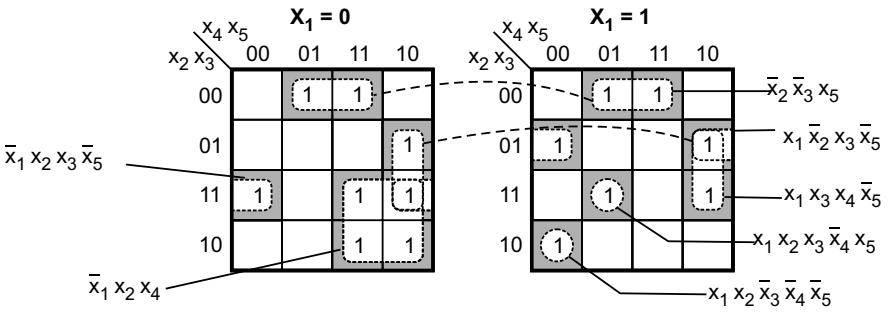


Fig. 1.8.6

Example 1.8.5 *Simplify the five variable switching function*

$f_{(EDCBA)} = \Sigma\, m\ (3, 5, 6, 8, 9, 12, 13, 14, 19, 22, 24, 25, 30).$ **Dec.-03**

Solution :

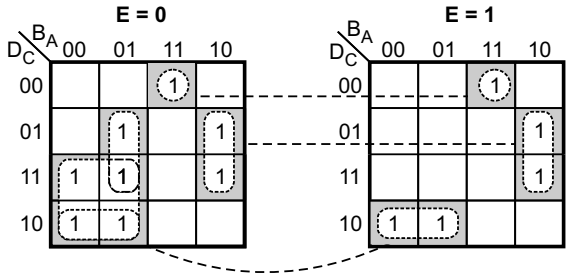


Fig. 1.8.7

$$f(E,D,C,B,A) = \overline{D}\,\overline{C}\,B\,A + \overline{E}\,C\,\overline{B}\,A + C\,B\,\overline{A} + \overline{E}\,D\,\overline{B} + D\,\overline{C}\,\overline{B}$$

Examples for Practice

Example 1.8.6 *Use the K-map method to simplify the following 5 variable function*
$F = \sum (3,6,7,8,10,12,14,17,19,20,21,24,25,27,31)$

Ans. : F = $\overline{A}\,\overline{B}$ DE + $\overline{A}\,\overline{B}$ CD + $\overline{A}$ B $\overline{E}$ + A $\overline{C}$ E + A$\overline{B}\overline{C}\overline{D}$ + ABDE + AB$\overline{C}$ $\overline{D}$

Example 1.8.7 *Apply branching method to simplify the following function.*
$F(A,B,C,D,E) = \pi M(0,1,4,5,9,11,13,15,16,17,25,27,28,29,31)$
$d(20,21,22,30)$. **Ans. : F (A, B, C, D, E) = (B + D) ($\overline{A}$ + $\overline{B}$ + $\overline{C}$) ($\overline{B}$ + $\overline{E}$)**

Example 1.8.8 *Simplify the following Boolean function using K Map.* **Dec.-11, Marks 8**
F (A, B, C, D, E) = $\sum$ (0, 1, 4, 5, 16, 17, 21, 25, 29)

Ans. : $\overline{A}\overline{B}\overline{D}$ + $\overline{B}\overline{C}\overline{D}$ + A$\overline{D}$E

1.9 Summary of Rules for K-Map Simplification

Rules for Simplifying logic function using K-map are :

1. Group should not include any cell containing a zero.
2. The number of cells in a group must be a power of 2, such as 1, 2, 4, 8 or 16.
3. Group may be horizontal, vertical but not diagonal.
4. Cell containing 1 must be included in at least one group.
5. Groups may overlap.
6. Each group should be as large as possible to get maximum simplification.
7. Groups may be wrapped around the map. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.
8. A cell may be grouped more than once. The only condition is that every group must have at least one cell that does not belong to any other group. Otherwise, redundant terms will result.
9. We need not group all don't care cells, only those that actually contribute to a maximum simplification.
10. All above rules are stated considering the SOP simplification. In case of POS simplification all rules are same except 0 (zero) takes place of 1 (one).

Review Question

1. *State the rules for K-map simplification.*

1.10 Limitations of Karnaugh Map

The map method of simplification is convenient as long as the number of variables does not exceed five or six. As the number of variables increases it is difficult to make judgements about which combinations form the minimum expression. In case of complex problem with 7, 8 or even 10 variables it is almost an impossible task to simplify expression by the mapping method. Another important point is that the K-map simplification is manual technique and simplification process is heavily depends on the human abilities. To meet this need, W. V. Quine and E. J. McCluskey developed an exact tabular method to simplify the Boolean expression. This method is called the **Quine McCluskey** or **tabular method.**

Review Question

1. *Explain the limitations of Karnaugh map.*

1.11 Quine-McCluskey Method of Minimization

Dec.-06, 09, 08, 10, 12, May-07, 09, 10, 12

In simplification of Boolean expression we observed that the adjacent minterms can be reduced. These minterms are reduced because they differ by only one literal. For example, $A B \overline{C}$ and $A \overline{B} \overline{C}$ can be reduced because only the B literal differs. The binary numbers equivalent to these minterms are 110 and 100. Note that the 2's place indicates precisely the same information that the literal represented, namely, that the B is the only literal that differs. Thus we can say that the minterms whose binary equivalent differ only in one place can be combined to reduce the minterms. This is the fundamental principle of the Quine McCluskey method.

Like in other methods, in this method minterms are listed to specify the given function. However, the minterms are written in their binary equivalents. These minterms are grouped according to number of 1s contained and separated by a horizontal line between each number of 1s category, as shown in the Table 1.11.1 (column (b)). This separation of minterms helps in searching the binary minterms that differ only in one place. Once the separation is over, each binary number is compared with every term in the next higher category and if they differ by only one position, a check mark is placed beside each of the two terms and then the term is copied in the second column with a '–' in the position that they differed.

For example, if the terms are 0000 and 0010 then the resultant term will be 0 0 – 0. This term, $0\,0-0$ is called an **implicant**, it implies the $\overline{A}\,\overline{B}\,\overline{D}$ could be a term in the final expression, covering minterms $\overline{A}\,B\,\overline{C}\,\overline{D}$ and $\overline{A}\,B\,C\,\overline{D}$. This process of comparison is repeated for every minterm. Once this process is completed the same process is applied to the new resultant terms which are placed in the Table 1.11.2 column (c). These cycles

are continued until a single pass through a cycle yields no further elimination of literals. The remaining terms and all the terms that did not match during the process are called the **prime implicants**. Summing one or more prime implicant gives the simplified Boolean expression.

1.11.1 Algorithm for Generating Prime Implicants

1. List all minterms in the binary form.
2. Arrange the minterms according to number of 1s.
3. Compare each binary number with every term in the adjacent next higher category and if they **differ only by one position**, put a check mark and copy the term in the next column with '–' in the position that they differed.
4. Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.
5. List all prime implicants.
6. Select the minimum number of prime implicants which must cover all the minterms.

See the following examples to illustrate the algorithm.

Examples for Understanding

Example 1.11.1 *Simplify the following Boolean function by using a Quine McCluskey method.* $F(A, B, C, D) = \sum m(0, 2, 3, 6, 7, 8, 10, 12, 13)$. **Dec.-09, Marks 16**

Solution :

Step 1 : List all minterms in the binary form as shown in Table 1.11.1 (column (a)).

Step 2 : Arrange the minterms according to number of 1s, as shown in the Table 1.11.1 (column (b)).

Minterm	Binary representation	Minterm	Binary representation
m_0	0 0 0 0	m_0	0 0 0 0 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	m_8	1 0 0 0 ✓
m_6	0 1 1 0	m_3	0 0 1 1 ✓
m_7	0 1 1 1	m_6	0 1 1 0 ✓

m_8	1 0 0 0	m_{10}	1 0 1 0	✓
m_{10}	1 0 1 0	m_{12}	1 1 0 0	✓
m_{12}	1 1 0 0	m_7	0 1 1 1	✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1	✓
Column (a)		Column (b)		

Table 1.11.1

Step 3 : Compare each binary number with every term in the adjacent next higher category and if they **differ only by one position**, put a check mark and copy the term in the next column with '–' in the position that they differed.

Step 4 : Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation		Minterms	Binary representation
0, 2	0 0 – 0	✓	0, 2, 8, 10	– 0 – 0
0, 8	– 0 0 0	✓	2, 3, 6, 7	0 – 1 –
2, 3	0 0 1 –	✓		
2, 6	0 – 1 0	✓		
2, 10	– 0 1 0	✓		
8, 10	1 0 – 0	✓		
8, 12	1 – 0 0			
3, 7	0 – 1 1	✓		
6, 7	0 1 1 –	✓		
12, 13	1 1 0 –			
Column (c)			Column (d)	

Table 1.11.2

Step 5 : List the prime implicants.

Prime implicants		Binary representation
$A\bar{C}\bar{D}$	8, 12	1 – 0 0
$AB\bar{C}$	12, 13	1 1 0 –
$\bar{B}\bar{D}$	0, 2, 8, 10	– 0 – 0
$\bar{A}C$	2, 3, 6, 7	0 – 1 –

Table 1.11.3

Step 6 : Select the minimum number of prime implicants which must cover all the minterms.

Prime implicants		m_0 (Col 1)	m_2 (Col 2)	m_3 (Col 3)	m_6 (Col 4)	m_7 (Col 5)	m_8 (Col 6)	m_{10} (Col 7)	m_{12} (Col 8)	m_{13} (Col 9)
$A\overline{C}\,\overline{D}$	8, 12						•		•	
$AB\overline{C}$	12, 13 ✓								⊙	⊙
$\overline{B}\,\overline{D}$	0, 2, 8, 10 ✓	⊙	⊙				⊙	⊙		
$\overline{A}C$	2, 3, 6, 7 ✓		⊙	⊙	⊙	⊙				

Table 1.11.4 Prime implicant selection chart

Table 1.11.4 shows prime implicant selection chart. Each prime implicant is represented in a row and each minterm in a column. Dots are placed in each row to show the composition of minterms that make the prime implicants. From this chart we have to select the minimum number of prime implicants which must cover all the minterms. The selection procedure is as follows.

Search for single dot columns and select the prime implicants corresponding to that dot by putting the check mark in front of it.

Search for multiple dot columns one by one. If the corresponding minterm is already included in the final expression ignore the minterm and go to next multi-dot column ; otherwise include the corresponding prime implicant in the final expression.

In our case, column 1 (m_0) has single dot, so include corresponding prime implicant (0, 2, 8, 10). Column 3 (m_3) has single dot so include corresponding prime implicant (2, 3, 6, 7). Columns 4, 5 and 7 have single dots but the corresponding minterms are already included in the final expression.

Column 9 has single dot so include corresponding prime implicant (12, 13) in the final expression. Now search for multi-dot columns. Columns 2, 6 and 8 have multi-dots but their minterms are already included in the final expression. Therefore, the final expression is

$$F(A, B, C, D) = (1\ 1\ 0\ -) + (-\ 0\ -\ 0) + (0\ -\ 1\ -)$$
$$= AB\overline{C} + \overline{B}\,\overline{D} + \overline{A}C$$

Example 1.11.2 *Minimize the expression using Quine McCluskey method.*
$Y = \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,\overline{B}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C\,\overline{D}$.

Solution :

Step 1 : List all minterms in the binary form, as shown in Table 1.11.5 (column (a)).

Step 2 : Arrange minterms according to categories of 1s, as shown in the Table 1.11.5 (column (b)) .

Minterm	Binary representation	Minterm	Binary representation
m_4	0 1 0 0	m_2	0 0 1 0
m_5	0 1 0 1	m_4	0 1 0 0 ✓
m_{12}	1 1 0 0	m_5	0 1 0 1 ✓
m_{13}	1 1 0 1	m_9	1 0 0 1 ✓
m_9	1 0 0 1	m_{12}	1 1 0 0 ✓
m_2	0 0 1 0	m_{13}	1 1 0 1 ✓
Column (a)		Column (b)	

Table 1.11.5

Step 3 : Compare each binary number with every term in the next higher category and if they differ by only one position put a check mark and copy the term in the next column with '–' in the position that they differed.

Minterms	Binary representation	Minterms	Binary representation
4, 5	0 1 0 – ✓	4, 5, 12, 13	– 1 0 –
4, 12	– 1 0 0 ✓		
5, 13	– 1 0 1 ✓		
9, 13	1 – 0 1		
12, 13	1 1 0 – ✓		

Table 1.11.6

Step 4 : Apply the same process described in step 3 for the resultant column and continue this cycles until a single pass through cycle yields no further elimination of literals.

Step 5 : List the prime implicants.

Prime implicants		Binary representation
$\overline{A}\ \overline{B}\ C\ \overline{D}$	2	0 0 1 0
$A\ \overline{C}\ D$	9, 13	1 – 0 1
$B\ \overline{C}$	4, 5, 12, 13	– 1 0 –

Table 1.11.7

Step 6 : Select the minimum number of prime implicants which must cover all the minterms.

Prime implicants		m_2 (Col 1)	m_4 (Col 2)	m_5 (Col 3)	m_9 (Col 4)	m_{10} (Col 7)	m_{12} (Col 5)	m_{13} (Col 9)
$\overline{A}\,\overline{B}\,C\,\overline{D}$	2 ✓	⊙						
$A\,\overline{C}\,D$	9, 13 ✓				⊙		⊙	⊙
$B\,\overline{C}$	4, 5, 12, 13 ✓		⊙	⊙		⊙	⊙	⊙

Table 1.11.8 Prime implicant selection chart

The final expression is $Y = (0\ 0\ 1\ 0) + (1\ -\ 0\ 1) + (-\ 1\ 0\ -) = \overline{A}\,\overline{B}C\,\overline{D} + A\,\overline{C}\,D + B\overline{C}$

1.11.2 Quine McCluskey using Don't Care Terms

The same rules that applied to using don't care terms with Karnaugh map are appropriate for Quine McCluskey. Here, don't care terms are never included as prime implicants by **themselves.** Let us consider the following example.

Example for Understanding

Example 1.11.3 *Simplify the following using tabulation methods.*
$Y(w, x, y, z) = \sum m\,(1, 2, 3, 5, 9, 12, 14, 15) + \sum d\,(4, 8, 11).$

Solution : It is important to note that don't care conditions are used to find the prime implicant but it is not compulsory to include don't care terms in the final expression.

Step 1 : List all minterms in the binary form, as shown in the Table 1.11.9 (Column (a)).

Minterm	Binary representation	Minterm	Binary representation
m_1	0 0 0 1	m_1	0 0 0 1 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_3	0 0 1 1	m_4	0 1 0 0 ✓
m_5	0 1 0 1	m_8	1 0 0 0 ✓
m_9	1 0 0 1	m_3	0 0 1 1 ✓
m_{12}	1 1 0 0	m_5	0 1 0 1 ✓
m_{14}	1 1 1 0	m_9	1 0 0 1 ✓
m_{15}	1 1 1 1	m_{12}	1 1 0 0 ✓
dm_4	0 1 0 0	m_{11}	1 0 1 1 ✓
dm_8	1 0 0 0	m_{14}	1 1 1 0 ✓
dm_{11}	1 0 1 1	m_{15}	1 1 1 1 ✓
Column (a)		Column (b)	

Table 1.11.9

Step 2 : Arrange the minterms according to categories of 1s as shown in the Table 1.11.9 (Column (b)).

Step 3 : Compare each binary number with every term in the adjacent next higher category and if they differ by only one position put a check mark and copy the term in the next column with '–' in the position that they differed.

Step 4 : Apply the same process described in step 3 for the resultant column and continue these cycles until a single pass through cycle yields no further elimination of literals.

Minterms	Binary representation					Minterms	Binary representation
1, 3	0	0	–	1	✓	1, 3, 9, 11	– 0 – 1
1, 5	0	–	0	1			
1, 9	–	0	0	1	✓		
2, 3	0	0	1	–			
4, 5	0	1	0	–			
4, 12	–	1	0	0			
8, 9	1	0	0	–			
8, 12	1	–	0	0			
3, 11	–	0	1	1	✓		
9, 11	1	0	–	1	✓		
12, 14	1	1	–	0			
11, 15	1	–	1	1			
14, 15	1	1	1	–			

Table 1.11.10

Step 5 : List the prime implicants.

Prime implicants		Binary representation			
$\overline{A}\ \overline{C}\ D$	1, 5	0	–	0	1
$\overline{A}\ \overline{B}\ C$	2, 3	0	0	1	–
$\overline{A}\ B\ \overline{C}$	4, 5	0	1	0	–
$B\ \overline{C}\ \overline{D}$	4, 12	–	1	0	0
$A\ \overline{B}\ \overline{C}$	8, 9	1	0	0	–
$A\ \overline{C}\ \overline{D}$	8, 12	1	–	0	0
$A\ B\ \overline{D}$	12, 14	1	1	–	0
A C D	11, 15	1	–	1	1
A B C	14, 15	1	1	1	–
$\overline{B}\ D$	1, 3, 9, 11	–	0	–	1

Table 1.11.11

Step 6 : Select the minimum number of prime implicants which must cover all the minterms, except don't care minterms.

Prime Implicants		m_1 (Col 1)	m_2 (Col 2)	m_3 (Col 3)	dm_4 (Col 4)	m_5 (Col 5)	dm_8 (Col 6)	m_9 (Col 7)	dm_{11} (Col 8)	m_{12} (Col 9)	m_{14} (Col10)	m_{15} (Col11)
$\overline{A}\,\overline{C}\,D$	1, 5 ✓	⊙				⊙						
$\overline{A}\,\overline{B}\,C$	2, 3 ✓		⊙	⊙								
$\overline{A}\,B\,\overline{C}$	4, 5				•	•						
$B\,\overline{C}\,\overline{D}$	4, 12				•					•		
$A\,\overline{B}\,\overline{C}$	8, 9						•	•				
$A\,\overline{C}\,\overline{D}$	8, 12						•			•		
$A\,B\,\overline{D}$	12, 14 ✓									⊙	⊙	
$A\,C\,D$	11, 15								•			•
$A\,B\,C$	14, 15 ✓										⊙	⊙
$\overline{B}\,D$	1, 3, 9, 11 ✓	⊙		⊙				⊙	⊙			

Essential prime implicant

Table 1.11.12 Prime implicant selection table

Only column 2 has single dot i.e. it is essential prime implicant and hence the prime implicant corresponding to it ($m_{2,3}$) is included in the final expression. Now search for multi-dot columns. Column 1 has multi-dot and the corresponding terms are not included in the final expression, so we can include either $m_{1,5}$ or $m_{1,3,9,11}$. Let us include prime implicant which has more minterms. Thus minterm 1, 3, 9, 11 is included. Now minterm for column 3 is already included and minterm for column 4 is don't care. Column 5 has a minterm which is not included yet, therefore prime implicant 1, 5 is included in the final expression. The minterms from column 6 and 8 are don't care and the minterm of column 7 (m_9) is already included in the final expression. Column 9 has minterm 12 which is not included yet.

To include this term we have 3 options. We include prime implicant 12, 14 because with this inclusion we can also cover minterm 14 in the final expression. The minterm from the remaining column 11 (m_{15}) can be included in the final expression by including prime implicant 14, 15. Therefore, the final expression is

$$Y = (0 - 0\ 1) + (0\ 0\ 1 -) + (1\ 1 - 0) + (1\ 1\ 1 -) + (- 0 - 1)$$

$$= \overline{A}\,\overline{C}\,D + \overline{A}\,\overline{B}\,C + A\,B\,\overline{D} + A\,B\,C + \overline{B}\,D$$

1.11.3 Prime Implicant Table and Redundant Prime Implicants

We have seen that the prime implicant selection table displays pictorially the covering relationships between the prime implicants and the minterms of the function. It consists of an array of u columns and v rows, where u and v designate the number of minterms for which the function takes on the value 1 and the number of prime implicants, respectively. The entries of the i^{th} row in the chart consists of dots placed at its intersections with the columns, corresponding to minterms covered by the i^{th} prime implicant. For example, the prime implicant chart of f(ABC) = $\sum$ m(0, 1, 2, 5, 6, 7) is shown in Table 1.11.13 (b). It consists of 6 columns corresponding to the minterms of f and six rows which correspond to the prime implicant generated in Table 1.11.13 (a).

Minterm	Binary representation	Minterms	Binary representation
0	0 0 0	0, 1	0 0 –
1	0 0 1	0, 2	0 – 0
2	0 1 0	1, 5	– 0 1
5	1 0 1	2, 6	– 1 0
6	1 1 0	5, 7	1 – 1
7	1 1 1	6, 7	1 1 –

Table 1.11.13 (a)

Prime implicants		m_0	m_1	m_2	m_5	m_6	m_7
$\overline{A}\,\overline{B}$	0, 1 ✓	⊙	⊙				
$\overline{A}\,\overline{C}$	0, 2	•		•			
$\overline{B}\,C$	1, 5		•		•		
$B\,\overline{C}$	2, 6 ✓			⊙		⊙	
$A\,C$	5, 7 ✓				⊙		⊙
$A\,B$	6, 7					•	•

Table 1.11.13 (b) Prime implicant selection table

The problem now is to select a minimal subset of prime implicants such that column contains at least one dot in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected as small as possible. In our example, each column has two dots. It is a special case. When each column in the prime implicant selection chart has two or more dots, the chart is known as **cyclic prime implicant table.**

Since all columns have two dots, we have to proceed by trial and error. Both (0, 1) and (0, 2) cover column 0, so we will try (0, 1). After selecting row (0, 1) and columns 0, and 1 we examine column 2, which is covered by (0, 2) and (2, 6). The best choice is (2, 6) because it covers two of the remaining columns while (0, 2) covers only one of the remaining columns. After selecting row (2, 6) and columns 2 and 6, we see that (5, 7) covers the remaining columns and completes the solution. Therefore, one of the solution is $f = \overline{A}\,\overline{B} + B\,\overline{C} + A\,C$. The prime implicants which are not selected to get the solution [(0, 2), (1, 5), (6, 7)] are called **redundant prime implicants**.

1.11.4 Advantages and Disadvantages of Quine McCluskey Method

The Quine McCluskey method of simplification can be applied to any number of variables and is algorithmic in nature. The simplification process of Quine-McCluskey becomes lengthy and time consuming as number of variables increase; however the algorithm can be easily programmed to run on a computer to identify prime implicants and implicates of any function.

Examples with Solutions

Example 1.11.4 *Obtain the minimum SOP using Quine McCluskey's method and verify using K-map.*

$F = m_0 + m_2 + m_4 + m_8 + m_9 + m_{10} + m_{11} + m_{12} + m_{13}$

Dec.-06, 10, May-07, 09, Marks 8

Solution :

Minterm	Primary Representation	Minterm	Binary Representation
m_0	0 0 0 0	m_0	0 0 0 0 ✓
m_2	0 0 1 0	m_2	0 0 1 0 ✓
m_4	0 1 0 0	m_4	0 1 0 0 ✓
m_8	1 0 0 0	m_8	1 0 0 0 ✓
m_9	1 0 0 1	m_9	1 0 0 1 ✓
m_{10}	1 0 1 0	m_{10}	1 0 1 0 ✓
m_{11}	1 0 1 1	m_{12}	1 1 0 0 ✓
m_{12}	1 1 0 0	m_{11}	1 0 1 1 ✓
m_{13}	1 1 0 1	m_{13}	1 1 0 1 ✓

Minterm	Binary Representation		Minterm	Binary Representation
✓ 0, 2	0 –	0 0	0, 2, 8, 10	– 0 – 0
✓ 0, 4	0 0	– 0	0, 8, 4, 12	– – 0 0
✓ 0, 8	– 0	0 0		
✓ 2, 10	– 1	0 0	8, 9, 10, 11	1 0 – –
✓ 4, 12	– 0	1 0	8, 9, 12, 13	1 – 0 –
✓ 8, 9	1 0	0 –		
✓ 8, 12	– 0	1 0		
✓ 8, 10	1 –	0 0		
✓ 9, 11	1 –	0 1		
✓ 9, 13	1 0	– 1		
✓ 10, 11	1 1	0 –		
✓ 12, 13	1 0	1 –		

Prime Implicants		Binary Representation
9, 13	$A\overline{C}D$	1 0 – 1
8, 9, 12, 13	$A\overline{C}$	1 – 0 –
0, 2, 8, 10	$\overline{B}\overline{D}$	– 0 – 0
0, 8, 4, 12	$\overline{C}\overline{D}$	– – 0 0
8, 9, 10, 11	$A\overline{B}$	1 0 – –

$$F(A, B, C, D) = A\overline{C} + \overline{B}\,\overline{D} + \overline{C}\,\overline{D} + A\overline{B}$$

Prime implicants		m_0	m_2	m_4	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}
$A\overline{C}D$	9, 13					.				.
$A\overline{C}$	8, 9, 12, 13 ✓				⊙	⊙			⊙	⊙
$\overline{B}\,\overline{D}$	0, 2, 8, 10 ✓	⊙	⊙		⊙		⊙			
$\overline{C}\,\overline{D}$	0, 8, 4, 12 ✓	⊙		⊙	⊙				⊙	
$A\overline{B}$	8, 9, 10, 11				⊙	⊙	⊙	⊙		

$$F(A, B, C, D) = \overline{C}\,\overline{D} + A\,\overline{C} + A\,\overline{B} + \overline{B}\,\overline{D}$$

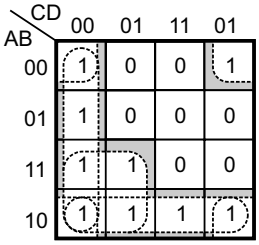


Fig. 1.11.1

Example 1.11.5 *Determine the prime implicants of the following function and verify using K map.* $F = (A, B, C, D) = \sum (3, 4, 5, 7, 9, 13, 14, 15)$ **May-09, Marks 16**

Solution :

Minterms	Binary representation	Minterms	Binary representation
m_3	0 0 1 1	m_4 ✓	0 1 0 0
m_4	0 1 0 0	m_3 ✓	0 0 1 1
m_5	0 1 0 1	m_5 ✓	0 1 0 1
m_7	0 1 1 1	m_9 ✓	1 0 0 1
m_9	1 0 0 1	m_7 ✓	0 1 1 1
m_{13}	1 1 0 1	m_{13} ✓	1 1 0 1
m_{14}	1 1 1 0	m_{14} ✓	1 1 1 0
m_{15}	1 1 1 1	m_{15} ✓	1 1 1 1

Minterms	Binary representation	Minterms	Binary representation
4, 5	0 1 0 –		
3, 7	0 – 1 1	5, 7, 13, 15	– 1 – 1
5, 7 ✓	0 1 – 1		
5, 13 ✓	– 1 0 1		
9, 13	1 – 0 1		
7, 15 ✓	– 1 1 1		
13, 15 ✓	1 1 – 1		
14, 15	1 1 1 –		

Prime implicants

Minterms	Binary representation			
4, 5	0	1 –	0	$\overline{A}B\overline{C}$
3, 7	0	– 1	1	$\overline{A}CD$
9, 13	1	– 1	0	$A\overline{C}D$
14, 15	1	1 –	1	A B C
5, 7, 13, 15	–	1 1	–	BD

Verifying prime implicants using K-map

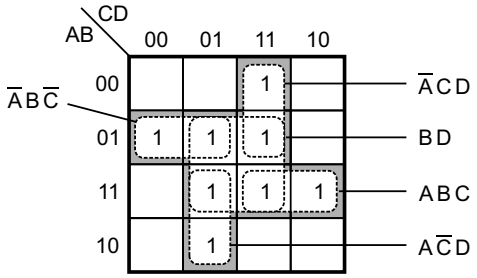


Fig. 1.11.2

Prime implicants are verified. The essential prime implicants are,

$$F(A, B, C, D) = \overline{A}B\overline{C} + \overline{A}CD + A\overline{C}D + ABC$$

Example 1.11.6 *Simplify the following fuction using Quine Mc Clusky method.*

$F(A, B, C, D, E, F, G) = \sum (20, 28, 52, 60)$ **Dec.-10, Marks 10**

Solution :

Minterms	Binary Representation							Minterms	Binary Representation						
m_{20}	0	0	1	0	1	0	0	20, 28 ✓	0	0	1	–	1	0	0
m_{28}	0	0	1	1	1	0	0	20, 52 ✓	0	–	1	0	1	0	0
m_{52}	0	1	1	0	1	0	0	28, 60 ✓	0	–	1	1	1	0	0
m_{60}	0	1	1	1	1	0	0	52, 60 ✓	0	1	1	–	1	0	0

Minterms	Binary Representation
20, 28, 52, 60	0 – 1 – 1 0 0

$\therefore$ $F(A, B, C, D, E, F, G) = \overline{A}\ C\ E\ \overline{F}\ \overline{G}$

Examples for Practice

Example 1.11.7 *Find prime implicants for the Boolean expression by using Quine McCluskey method. f(A,B,C,D) = $\sum$ (1, 3, 6, 7, 8, 9, 10, 12, 14, 15) + d(11, 13)*

Ans. : F(A, B, C, D) = $\overline{B}D + A + BC$

Example 1.11.8 *Give simplified logic equation using Quine-McCluskey method for the following Boolean function f(A, B, C, D) = Σ m (0, 1, 2, 3, 10, 11, 12, 13, 14, 15)*

Ans. : F(A, B, C, D) = $\overline{A}\,\overline{B} + AC + AB$

Example 1.11.9 *Minimize the following using Quine-McCluskey method.*
Σ m (0, 1, 2, 8, 9, 15, 17, 21, 24, 25, 27, 31) **Dec.-8, Marks 7**

Ans. : $\overline{A}\,\overline{C}\,\overline{D} + B\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,\overline{E} + A\,B\,\overline{C}\,E + BCDE + A\,\overline{B}\,\overline{D}\,E$

Example 1.11.10 *Minimize the given terms*
π M (0, 1, 4, 11, 13, 15) + π d (5, 7, 8) using Quine-McCluskey methods and verify the results using K-map methods. **May-10, Marks 12**

Ans. : f(A, B, C, D) = $\overline{A}\,C + A\,\overline{D} + A\,\overline{B}\,\overline{C}$

Example 1.11.11 *Using QM method simplify the Boolean expression*
$f(x_1, x_2, x_3, x_4, x_5) = \sum$ (0, 1, 4, 5, 16, 17, 21, 25, 29)
Dec.-10, Marks 16

Ans. : $f = \overline{x}_1\,\overline{x}_2\,\overline{x}_4 + \overline{x}_2\,\overline{x}_3\,\overline{x}_4 + x_1\,\overline{x}_4\,x_5$

Example 1.11.12 *Minimise the function, F using Quine Mclusky method.*
$F = \sum$(0, 1, 2, 8, 10, 11, 14, 15) **May-12, Marks 8**

Ans. : $F = AC + \overline{B}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}$

Example 1.11.13 *Simplify the given Boolean function using tabulation method.*
F(A, B, C, D) = $\sum$ m(1, 2, 3, 5, 7, 9, 10, 11, 13, 15). **Dec.-12, Marks 16**

Ans. : $D + \overline{B}\,C$

Review Questions

1. *Explain the Quine McCluskey method of simplification of Boolean function with the help of example.*
2. *State the algorithm for generating prime implicants.*
3. *What do you mean by cyclic prime implicants ?*
4. *State the advantages and disadvantages of Quine McCluskey method of simplification.*

1.12 Two Marks Questions with Answers

Q.1 ***What do you mean by literal ?***

Ans. : Definition : In Boolean Function, the total number of variables in complemented or uncomplemented form are called literals.

For example : The function F (A, B, C, D) = $A + \overline{B}\,\overline{C} + AC\overline{D}$ contains 4 literals.

Q.2 ***State two absorption properties of Boolean algebra.*** **Dec.-04**

Ans. : 1. A + AB = A. and 2. A (A + B) = A

Q.3 ***State the associative law of Boolean algebra.*** **May-08**

Ans. : Associative laws are as follows :

1. Law 1 (The associative law of addition) :

In the Oring of several variables, the result is the same regardless of the grouping of the variables. For three variables, A ORed with B OR C is the same as A OR B ORed with C.

i.e. A + (B + C) = (A + B) + C

2. Law 2 (The associative law of multiplication) :

It makes no difference in what order the variables are grouped when ANDing several variables. For three variables, A AND B ANDed with C is the same as A ANDed with B and C.

i.e. (AB) C = A(BC)

Q.4 ***Explain the De Morgan's theorem in Boolean Algebra.*** **May-11**

Ans. : Refer section 1.2.

Q.5 ***Explain the principle of duality with the help of example.***

Ans. : The duality theorem says that, starting with a Boolean relation, you can derive another Boolean relation by -

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign and
3. Complementing any 0 to 1 appearing in the expression.

For example : A + 0 = A. Using duality theorem, we can say that, $A \cdot 1 = A$.

Q.6 ***State and prove the consensus theorem in Boolean algebra.***

Ans. : Definition : In simplification of Boolean expression, the redundant term in an expression can be eliminated to form the equivalent expression. The theorem used for this simplification is called **consensus theorem**. For example, in expression of the $AB + \overline{A}C + BC$, the term BC is redundant and can be eliminated using consensus theorem.

Proof :

$$AB + \overline{A}C + BC = AB + \overline{A}C + \left(A + \overline{A}\right) BC$$

$$= AB + \overline{A}C + AB + \overline{A}C$$

$$= AB + \overline{A}C$$

Q.7 ***Name the two basic types of Boolean expressions.***

Ans. : There are two basic types of Boolean expressions are :

- Sum of product Form (SOP) and
- Product of Sum Form (POS)

Q.8 ***Name the two canonical forms for Boolean algebra.*** **May-07**

Ans. : Standard SOP and standard POS forms.

Q.9 ***Express $F = BC' + AC$ in a canonical SOP form.*** **Dec.-03**

Ans. :

$$F = B\overline{C} + A\,C$$

$$= \left(A + \overline{A}\right) B\,\overline{C} + A\,C\left(B + \overline{B}\right)$$

$$= A\,B\,\overline{C} + \overline{A}\,B\,\overline{C} + A\,B\,C + A\,\overline{B}\,C$$

Q.10 ***How many inputs are required for $W = A\overline{B}D + AC\overline{D} + EF$?*** **Dec.-04**

Ans. : 6-inputs.

Q.11 ***Simplify the following Boolean expressions to a minimum number of literals :***

a) $(X+Y)\left(X+\overline{Y}\right)$ *b)* $XY + \overline{X}Z + YZ$ **Dec.-05**

Ans. :

a) $(X+Y)\left(X+\overline{Y}\right) = XX + X\overline{Y} + XY + 0 = X + X\left(\overline{Y} + Y\right)$

$= X + X = X$

b) $XY + \overline{X}Z + YZ = XY\left(Z+\overline{Z}\right) + \overline{X}Z\left(Y+\overline{Y}\right) + \left(X+\overline{X}\right)YZ$

$= XYZ + XY\overline{Z} + \overline{X}YZ + \overline{X}\overline{Y}Z + XYZ + \overline{X}YZ$

$= XYZ + XY\overline{Z} + \overline{X}YZ + \overline{X}\overline{Y}Z$

$= XY\left(Z+\overline{Z}\right) + \overline{X}Z\left(Y+\overline{Y}\right) = XY + \overline{X}Z$

Q.12 ***Simplify the following Boolean expression.***
$a\overline{b}\,\overline{c} + a\overline{b}c + abc$. **May-05**

Ans. : $a\,\overline{b}\,\overline{c} + a\,\overline{b}\,c + abc = a\,\overline{b}\,\overline{c} + ac\,(\overline{b}+b)$

$= a\,\overline{b}\,\overline{c} + ac$

$= a\,(\overline{b}\,\overline{c} + c)$

$= a\,(\overline{b} + c)$ $\because A + \overline{A}B = A + B$

$= a\overline{b} + ac$

Q.13 *Simplify : x + x'y.* **Dec.-07, May-10**

Ans. : $z = x + \bar{x}y = x + xy + \bar{x}y$ $\because A + AB = A$

$= x + y(x + \bar{x})$ $\because x + \bar{x} = 1$

$= x + y$

Q.14 *Show that*

a) $a + a'b = a + b$ *b)* $x'y'z + x'yz + xy' = x'z + xy'$. **May-10**

Ans. : a) Refer example 1.2.1

b) $\bar{x}\,\bar{y}\,z + \bar{x}\,y\,z + x\,\bar{y}$

$= \bar{x}\,z(\bar{y} + y) + x\,\bar{y} = \bar{x}\,z + x\,\bar{y}$ $\because A + \bar{A} = A$

Q.15 *What is variable mapping ?* **May-07**

Ans. : Representing the minterms of the sum of product expression in the Karnaugh-map of appropriate variables is known as variable mapping.

Q.16 *What code is used to label the row headings and column heading of K-map and why ?*

Ans. :

- Gray code is used to label the rows and columns of K-map.
- In case of Gray code, only one variable changes between two consecutive number.This is useful in grouping pair, quad, or octets in k-maps and thus eliminating variables in the final expression. Hence, Gray code is used to label the rows and columns of K-map.

Q.17 *Show the Karnaugh map with the encircled groups for the Boolean function,* $F = \overline{C} + \overline{A}\,\overline{D} + A\,\overline{B}\,\overline{D}$. **May-06**

Ans. :

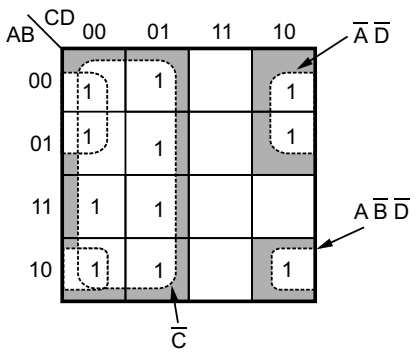


Fig. 1.12.1

Q.18 *What are don't care conditions and incompletely specified functions ?*

Ans. : Sometimes, while designing a digital circuit, certain conditions of input are of no use. Consider an example of BCD input for a seven segment display. In this case, inputs from 0 through 9 are valid inputs. Thus, rest of the inputs from 10 through 15 do not signify anything at the output side. The outputs for which inputs are not specified are called **don't care outputs**. The Boolean function in which don't care outputs exists is called incomplete Boolean function or incompletely specified functions.

Q.19 *Distinguish between completely specified function and incompletely specified function.* **May-09**

Ans. :

Sr. No.	Completely specified function	Incompletely specified function
1.	All input conditions occur.	Certain input conditions never occur.
2.	Each output corresponding to input conditions is defined.	Outputs corresponding to non-occurring input conditions are undefined.
3.	Each output has certain logical value, either 0 or 1.	Undefined output doesn't have certain logical value, but it is indicated by 'x' or 'd' in truth tables.
4.	In expression, $f(A, B, C) = \sum m(0, 1, 3, 5, 6)$ $= \pi M(1, 4, 7)$	In expression, $f(A, B, C) = \sum m(0, 1, 3) + d(5, 6)$ $= \pi M(2, 4, 7) + d(5, 6)$ Here outputs corresponding to terms 5 and 6 are assumed to be undefined ('x' or 'd') called as 'don't care'.

Q.20 *What are prime implicants ?* **Dec.-05, Dec.-08**

Ans. : All the implicants of a function determined using a Karnaugh map are called prime implicants.

Q.21 *State two significant features of tabular method of minimization of Boolean functions.* **Dec.-04**

Ans. :

1. It is an exact method to simplify the Boolean expression.
2. It can be used to simplify expression with 7, 8 or even 10 variables.

❑❑❑

Notes

2 Logic Gates

Syllabus

AND, OR, NOT, NAND, NOR, Exclusive-OR and Exclusive-NOR- Implementations of logic functions using gates, NAND-NOR implementations - Multilevel gate implementations - Multioutput gate implementations.

Contents

2.1 Logic Gates

2.2 Implementations of Logic Functions . . . . . **Dec.-06, 07,** · · · · · · · · · · · · · Marks 8

using Gates . . . . . **May-07, 12,** · · · · · · · · · · · · · Marks 12

2.3 Universal Gates . . . . . **May-05, 09,** · · · · · · · · · · · · · · Marks 4

. . . . . **Dec.-06,** · · · · · · · · · · · · · · · · Marks 2

2.4 NAND-NAND Implementation . . . . . **May-03, 05, 06, 09, 12,** · · · · Marks 16

. . . . . **Dec.-04, 05, 06, 07, 08, 10, 12,** Marks 8

2.5 NOR-NOR Implementation . . . . . **Dec.-04, 05, 07,** · · · · · · · · · · Marks 8

. . . . . **May-05, 10,** · · · · · · · · · · · · · Marks 16

2.6 Multilevel Gate Implementations

2.7 Multilevel NAND and NOR Implementations. . . . **Dec.-11,** · · · · · · · · · · · · · · · · Marks 6

2.8 Multiple Output Implementations

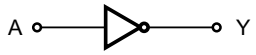
2.9 Two Marks Questions with Answers

2.1 Logic Gates

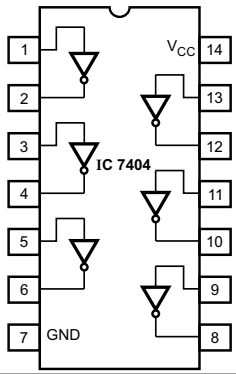
Logic gates are the basic elements that make up a digital system. The electronic gate is a circuit that is able to operate on a number of binary inputs in order to perform a particular logical function. The types of gates available are the NOT, AND, OR, NAND, NOR, exclusive-OR, and the exclusive-NOR.

NOT gate (Inverter) : The output is a complement of input.

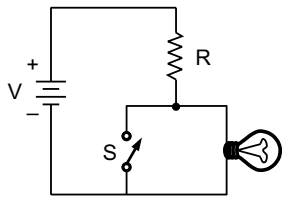
Logic Diagram (Symbol)



Pin Diagram



Switch Equivalent



Input	Output
Switch open (Low)	Lamp ON (High)
Switch close (High)	Lamp OFF (Low)

Application :

Used to complement (invert) digital signal.

Truth Table

Input	Output
A	**Y**
0	1
1	0

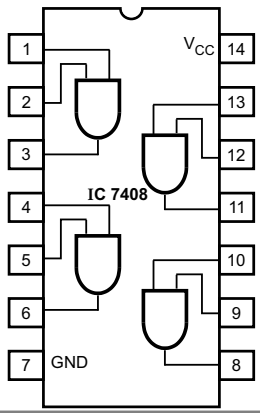
Boolean Expression

$$Y = \overline{A}$$

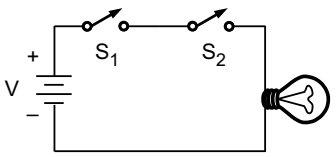
AND gate : The output is high only when all inputs are high.

Logic Diagram (Symbol)

Pin Diagram



Switch Equivalent



Input		Output
S_1	S_2	
Open (Low)	Open (Low)	Lamp OFF (Low)
Open (Low)	Close (High)	Lamp OFF (Low)
Close (High)	Open (Low)	Lamp OFF (Low)
Close (High)	Close (High)	Lamp ON (High)

Application : Used to implement logical AND operation.

Truth Table

Input		Output
A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

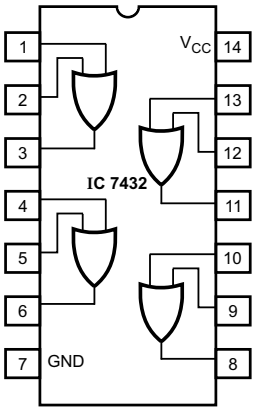
Boolean Expression

$$Y = A \cdot B$$

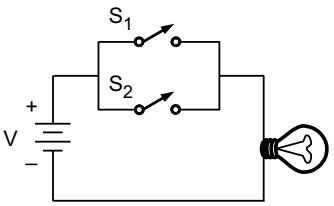
OR gate : The output is high when any of the inputs is high.

Logic Diagram (Symbol)

Pin Diagram



Switch Equivalent



Input		Output
S_1	S_2	
Open (Low)	Open (Low)	Lamp OFF (Low)
Open (Low)	Close (High)	Lamp ON (High)
Close (High)	Open (Low)	Lamp ON (High)
Close (High)	Close (High)	Lamp ON (High)

Application : Used to implement logical OR operation.

Truth Table

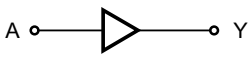
Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Expression

Y = A + B

Buffer : The output is same as input.

Symbol

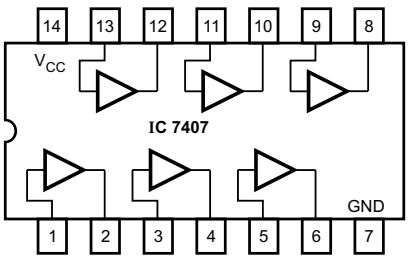


Application : It is used to increase output driving capacity.

Boolean Expression

Y = A

Pin Diagram



Truth Table

Input	Output
A	Y
0	0
1	1

NAND gate : The output is high only when one of the inputs is low.

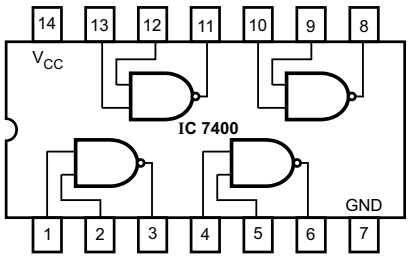
Symbol

Application : It can be used to implement any digital circuit.

Boolean Expression

$$Y = \overline{A \cdot B}$$

Pin Diagram



Truth Table

Input		Output
A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate : The output is high when all the inputs are low.

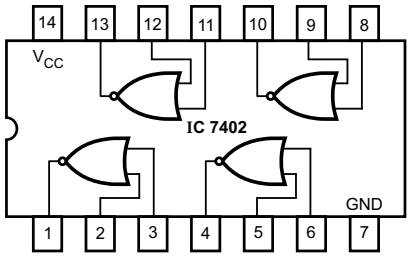
Symbol

Application : It can be used to implement any digital circuit.

Boolean Expression

$$Y = \overline{A + B}$$

Pin Diagram



Truth Table

Input		Output
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive OR (EX-OR) gate : The output is high only when odd number of inputs are high.

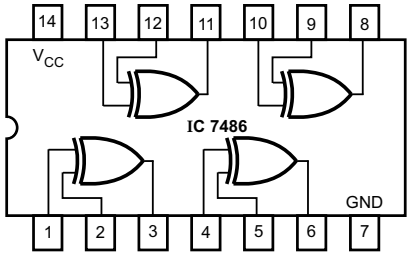
Symbol

Application : It is used to implement magnitude comparator, gray code converter, adder/subtractor circuits, parity generator, modulo-2 adder, etc.

Boolean Expression

$$Y = A \oplus B$$

Pin Diagram



Truth Table

Input		Output
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive NOR (EX-NOR) gate : The output is high only when even number of ones at the input or all inputs are high.

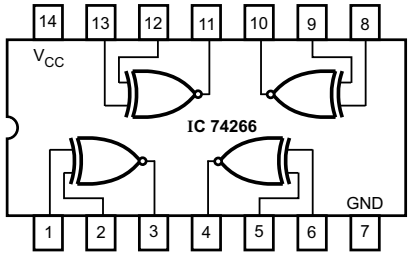
Symbol

Applications : It is used to implement even parity generator, comparator, even parity checker, etc.

Boolean Expression

$$Y = \overline{A \oplus B}$$

Pin Diagram



Truth Table

Input		Output
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	1

Wave-forms / Timing diagrams of all the gates

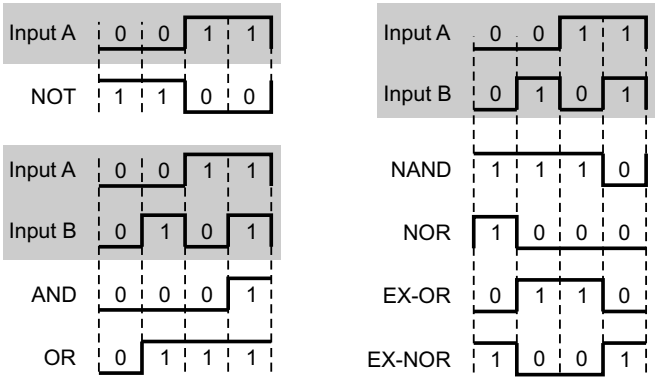

Input A 0 0 1 1
NOT 1 1 0 0
Input A 0 0 1 1
Input B 0 1 0 1
AND 0 0 0 1
OR 0 1 1 1
Input A 0 0 1 1
Input B 0 1 0 1
NAND 1 1 1 0
NOR 1 0 0 0
EX-OR 0 1 1 0
EX-NOR 1 0 0 1

Fig. 2.1.1

Note : EX-OR and EX-NOR gates are also known as **mutually exclusive gates.**

Review Question

1. Write the logic symbol, expression and truth table for the following logic gates :

1. (i) EX-OR (ii) NOR

2. (i) NAND (ii) EX-NOR

3. (i) NAND (ii) NOR (iii) EX-OR

2.2 Implementations of Logic Functions using Gates

Dec.-06, 07, May-07, 12

The Boolean algebra is used to express the output of any combinational network. Such a network can be implemented using logic gates. Let us see the implementation of SOP and POS Boolean expressions.

2.2.1 Implementation of SOP Boolean Expression

Consider the Boolean expression

$$F = AB + C\overline{D} + \overline{B}C$$

In this expression, we have three product terms with 2 literals in each product term. Thus, we can implement these product terms by using three 2-input AND gates, as shown in the Fig. 2.2.1. Expression tells that these product terms should be ORed to get the output F. We have three product terms so we have to use 3-input OR gate to obtain the sum of products. It is important to note that literals are complemented using NOT gates.

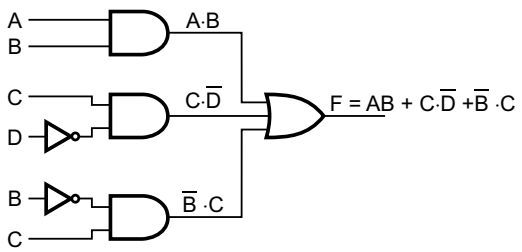


Fig. 2.2.1 Implementation of SOP Boolean expression

2.2.2 Implementation of POS Boolean Expression

Consider the Boolean expression

$$F = (A + B)\,(\overline{B} + C)\,(\overline{C} + D + E)$$

In this expression, we have three sum terms with 2 literals in two terms and 3 literals in one term. We can implement these sum terms by using two 2-input OR gates and one 3-input OR gate, as shown in the Fig. 2.2.2. Expression tells that these product terms should be ANDed to get the output F. We have three sum terms so we have to use 3-input AND gate to obtain the product of sums. It is important to note that literals are complemented using NOT gates.

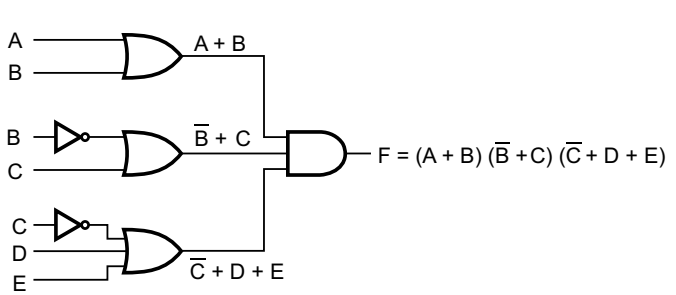


Fig. 2.2.2 Implementation of POS Boolean expression

In the previous examples we have seen that simplified SOP Boolean expression can be implemented using AND-OR gates. The AND-OR implementation is a two level implementation. In the first level we implement all product terms using AND gates and in the second level all product terms are logically ORed using OR gate. In case of POS expression we use OR-AND implementation. Here, we implement all sum terms using

OR gates in the first level and all sum terms are logically ANDed using AND gate, to get product of sum, in the second level. We know that, the logic gates are available in the integrated circuit (IC) packages. When we implement logic circuit using basic gates, we require ICs for AND, OR and NOT gates.

Many times it may happen that all gates from the IC packages are not required to build the circuit and thus remaining gates are unused. Consider a combinational circuit which requires two 2-input AND gates and one 2-input OR gate as shown in Fig. 2.2.3. To implement such a circuit we require IC 7408 (Four 2-input AND gates) and IC 7432 (Four 2-input OR gate). When we use these two ICs we find that two 2-input AND gates are unused and three 2-input OR gates are unused. Thus, the utility factor is very poor. This utility factor can be increased by using universal gates to implement logic functions.

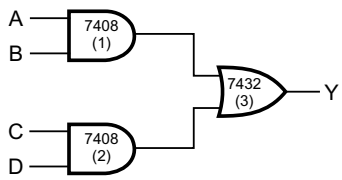


Fig. 2.2.3

Examples with Solutions

Example 2.2.1 *Implement the expression.*

1) AB + BCD + EFGH

2) (A + B) (C + D + E) (F + G + H + I) with logic gates.

Dec.-06, Marks 8

Solution :

1)

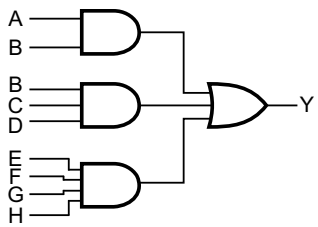


Fig. 2.2.4

2)

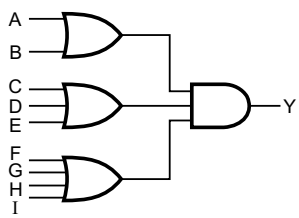


Fig. 2.2.5

Example 2.2.2 *Write the Boolean expression for the output of the system shown in Fig. 2.2.6 below :* **May-07, Marks 2**

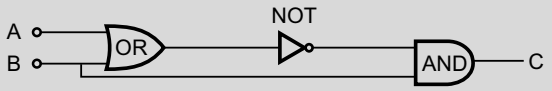


Fig. 2.2.6

Solution : $C = \left(\overline{A + B}\right) \cdot B = \overline{A} \cdot \overline{B} \cdot B = 0$

Example 2.2.3 *Draw the logic diagram for X = AB + B'C.* **Dec.-07, Marks 2**

Solution : $X = AB + \overline{B}C$

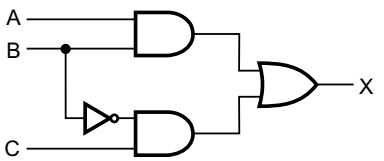


Fig. 2.2.7

Example 2.2.4 *Implement the Boolean expression using gates.*
X = (AB + C)' D + E. **Dec.-07, Marks 4**

Solution : $X = \overline{(AB + C)}\, D + E = \left(\overline{AB} \cdot \overline{C}\right) D + E = \overline{A}\,\overline{B}\,\overline{C}\,D + E$

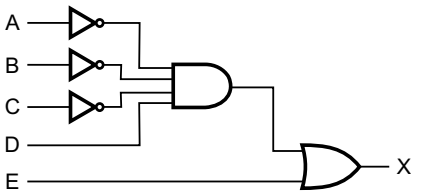
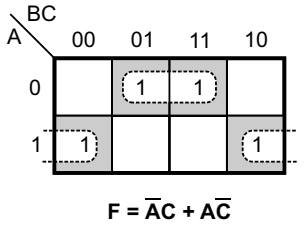


Fig. 2.2.8

Example 2.2.5 *Using Karnaugh map simplify the following expressions and implement using basic gates. 1)* $F = \sum(1,3,4,6)$ *2)* $F = \sum(1,3,7,11,15) + d(0,2,5)$. **May-12, Marks 12**

Solution : 1)



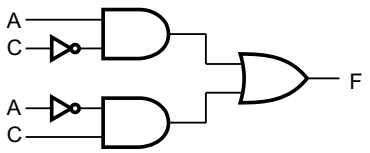
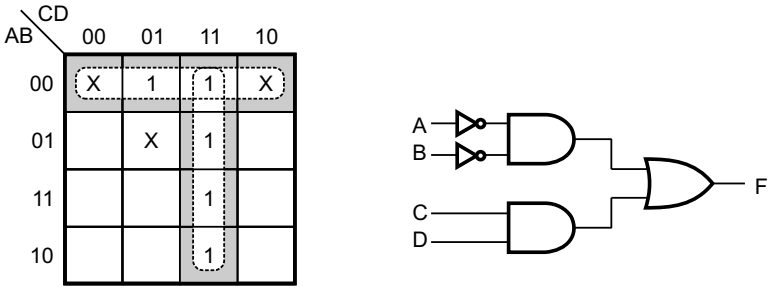


Fig. 2.2.9

2)



$F = \overline{A}\overline{B} + CD$

Fig. 2.2.10

2.3 Universal Gates

May-05, 09, Dec.-06

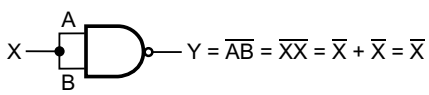
The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates.

2.3.1 NAND Gate

The NAND gate can be used to generate the NOT function, the AND function, the OR function, and the NOR function.

NOT Function :

An inverter can be made from a NAND gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 2.3.1, for a two-input gate.



	A	B	$\overline{AB}$	
X = 0	0	0	1	Y = 1
	0	1	1	
	1	0	1	
X = 1	1	1	0	Y = 0

Fig. 2.3.1 NOT function using NAND gate

AND Function :

An AND function can be generated using only NAND gates. It is generated by simply inverting output of NAND gate i.e. $\overline{\overline{AB}}$ = AB. Fig. 2.3.2 shows the two input AND gate using NAND gates.

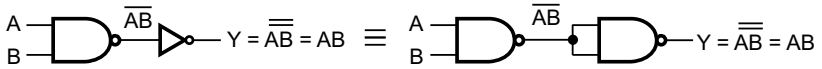


Fig. 2.3.2 AND function using NAND gates

A	B	AB
0	0	0
0	1	0
1	0	0
1	1	1

≡

A	B	$\overline{AB}$	$\overline{\overline{AB}}$
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2.3.1 Truth table

OR Function :

OR function is generated using only NAND gates as follows : We know that Boolean expression for OR gate is

$$Y = A + B$$
$$= \overline{\overline{A}} + \overline{\overline{B}} \quad \text{... } [\overline{\overline{A}} = A]$$
$$= \overline{\overline{A} \cdot \overline{B}} \quad \text{... DeMorgan's Theorem 1}$$

The above equation is implemented using only NAND gates as shown in the Fig. 2.3.3.

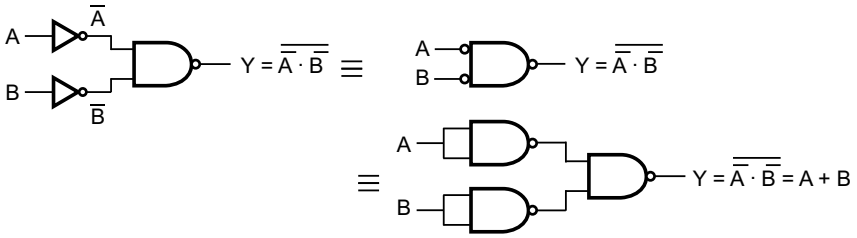


Fig. 2.3.3 OR function using only NAND gates

Note : Bubble at the input of NAND gate indicates inverted input.

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

≡

A	B	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Table 2.3.2 Truth table

NOR Function :

NOR function is generated using only NAND gates as follows : We know that Boolean expression for NOR gate is

$$Y = \overline{A + B}$$

$= \overline{A} \cdot \overline{B}$ ... DeMorgan's Theorem 2

$= \overline{\overline{A \cdot B}}$ ... $[\overline{\overline{A}} = A]$

The above equation is implemented using only NAND gates, as shown in the Fig. 2.3.4.

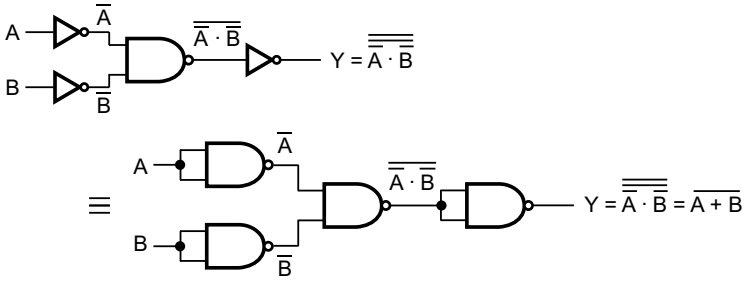


Fig. 2.3.4 NOR function using only NAND gates

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

≡

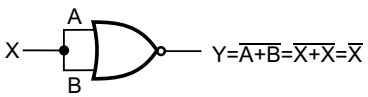
A	B	$\overline{A} \cdot \overline{B}$	$\overline{\overline{A} \cdot \overline{B}}$	$\overline{\overline{A} \cdot \overline{B}}$
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	1	0

2.3.2 NOR Gate

Similar to NAND gate, the NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

NOT Function :

An inverter can be made from a NOR gate by connecting all of the inputs together and creating, in effect, a single common input, as shown in Fig. 2.3.5.



	A	B	$\overline{A+B}$	
X = 0	0	0	1	Y = 1
	0	1	0	
	1	0	0	
X = 1	1	1	0	Y = 0

Fig. 2.3.5 NOT function using NOR gate

OR Function :

An OR function can be generated using only NOR gates. It can be generated by simply inverting output of NOR gate; i.e. $\overline{\overline{A+B}} = A + B$. Fig. 2.3.6 shows the two input OR gate using NOR gates.

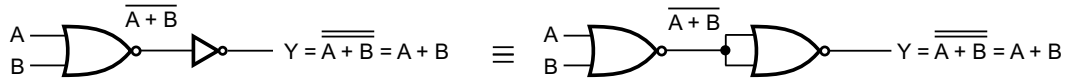


Fig. 2.3.6 OR function using NOR gates

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

≡

A	B	$\overline{A+B}$	$\overline{\overline{A+B}}$
0	0	1	0
0	1	0	1
1	0	0	1
1	1	0	1

Table 2.3.3 Truth table

AND Function :

AND function is generated using only NOR gates as follows : We know that Boolean expression for AND gate is

$$Y = A \cdot B$$

$$= \overline{\overline{A}} \cdot \overline{\overline{B}} \quad \text{... } [\overline{\overline{A}} = A]$$

$$= \overline{\overline{A} + \overline{B}} \quad \text{... De-Morgan's Theorem 2}$$

The above equation is implemented using only NOR gates as shown in the Fig. 2.3.7.

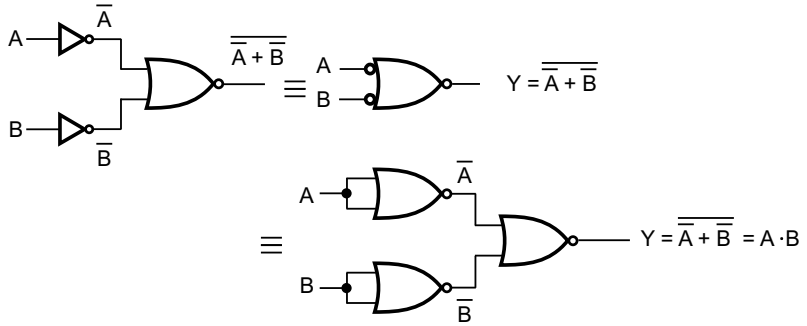


Fig. 2.3.7 AND function using NOR gates

Note : Bubble at the input of NOR gate indicates inverted input.

A	B	$A \cdot B$		A	B	$\overline{A} + \overline{B}$	$\overline{\overline{A} + \overline{B}}$
0	0	0		0	0	1	0
0	1	0	≡	0	1	1	0
1	0	0		1	0	1	0
1	1	1		1	1	0	1

Table 2.3.4 Truth table

NAND Function :

NAND function is generated using only NOR gates as follows : We know that Boolean expression for NAND gate is

$$Y = \overline{A \cdot B}$$

$$= \overline{A} + \overline{B} \quad \text{... DeMorgan's Theorem 1}$$

$$= \overline{\overline{\overline{A} + \overline{B}}} \quad \text{... } [\overline{\overline{A}} = A]$$

The above equation is implemented using only NOR gates, as shown in the Fig. 2.3.8.

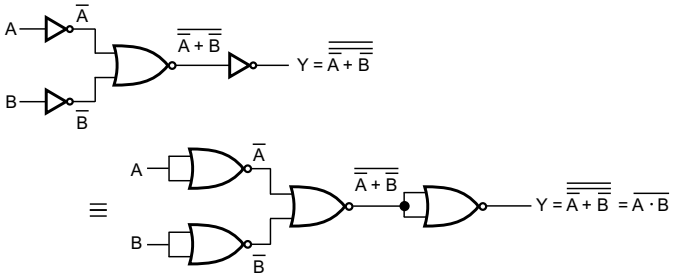


Fig. 2.3.8 NAND function using only NOR gates

A	B	$\overline{A \cdot B}$		A	B	$\overline{A} + \overline{B}$	$\overline{\overline{A} + \overline{B}}$	$\overline{\overline{\overline{A} + \overline{B}}}$
0	0	1		0	0	1	0	1
0	1	1	≡	0	1	1	0	1
1	0	1		1	0	1	0	1
1	1	0		1	1	0	1	0

Table 2.3.5 Truth table

Review Questions

1. *What are universal gates ? Give examples.*
2. *Why NAND and NOR gates are called universal gates ?*
3. *Why digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates ?* **Dec.-06, May-09, Marks 2**
4. *Realize (i) AND gate (ii) NOR gate using only NAND gates.*
5. *Realize (i) OR gate (ii) EX-OR gate using NAND gates.*
6. *Realize (i) OR gate (ii) AND gate using only NOR gates.* **May-05, Marks 4**

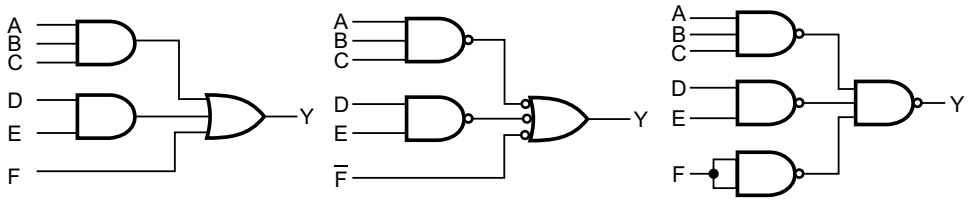
2.4 NAND-NAND Implementation

May-03, 05, 06, 09, 12, Dec.-04, 05, 06, 07, 08, 10, 12

The implementation of a Boolean function with NAND-NAND logic requires that the function be simplified in the sum of product form. The relationship between AND-OR logic and NAND-NAND logic is explained using following example.

Consider the Boolean function : $Y = A\ B\ C + D\ E + F$.

This Boolean function can be implemented using AND-OR logic, as shown in Fig. 2.4.1 (a).



(a) AND - OR (b) NAND - bubbled OR (c) NAND-NAND

Fig. 2.4.1 NAND-NAND implementation

Fig. 2.4.1 (b) shows the AND gates are replaced by NAND gates and the OR gate is replaced by a bubbled OR gate. The implementation shown in Fig. 2.4.1 (b) is equivalent to implementation shown in Fig. 2.4.1 (a), because two bubbled on the same line represent double inversion (complementation) which is equivalent to having no bubble on the line. In case of single variable, F, the complemented variable is again complemented by bubble to produce the normal value of F.

In Fig. 2.4.1 (c), the output NAND gate is redrawn with the conventional symbol. The NAND gate with same inputs gives complemented result, therefore $\overline{F}$ is replaced by NAND gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

From the above example we can summarize the rules for obtaining the NAND-NAND logic diagram from a Boolean function as follows :

1. Simplify the given Boolean function and express it in sum of product form (SOP form).
2. Draw a NAND gate for each product term of the function that has two or more literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.
3. If Boolean function includes any single literal or literals draw NAND gate for each single literal and connect corresponding literal as an input to the NAND gate.
4. Draw a single NAND gate in the second level, with inputs coming from outputs of first level gates.

Illustrative Examples

Example 2.4.1 *Implement the following Boolean function with NAND-NAND logic*

$Y = A\,C + A\,B\,C + \overline{A}\,B\,C + A\,B + D$

Solution : Step 1 : Simplify the given Boolean function.

$$Y = A\,C + A\,B\,C + \overline{A}\,B\,C + A\,B + D$$
$$= A\,C + B\,C\,(A + \overline{A}) + A\,B + D = A\,C + B\,C + A\,B + D$$

Step 2 : Implement using AND-OR logic.

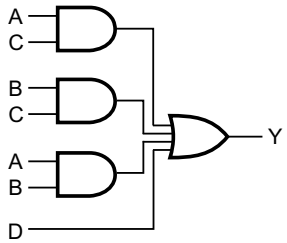


Fig. 2.4.2 (a)

Step 3 : Convert AND-OR logic to NAND-NAND logic.

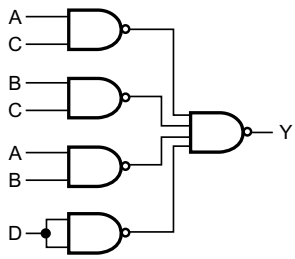


Fig. 2.4.2 (b)

Example 2.4.2 *Implement the following Boolean function with NAND-NAND logic*

$F = \overline{A}\,\overline{B} + \overline{A}\,C + \overline{B}\,C$

Solution :

Step 1 : Implement Boolean function with AND-OR logic.

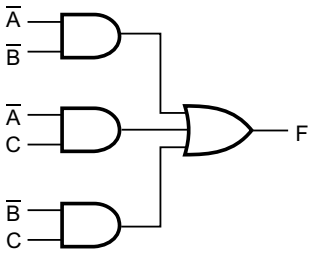


Fig. 2.4.3 (a)

Step 2 : Convert AND-OR logic to NAND-NAND logic.

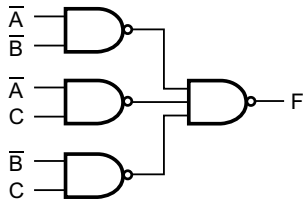


Fig. 2.4.3 (b)

Note : It is possible to directly go to step 2 skipping step 1. Here, step 1 is included for clear understanding.

Example 2.4.3 *Implement the following Boolean function with NAND-NAND logic.*
$F = (A, B, C) = \sum m\ (0, 1, 3, 5)$

Solution : **Step 1 :** Simplify the given Boolean function.

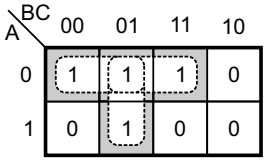


Fig. 2.4.4

$\therefore \quad F = \overline{A}\,\overline{B} + \overline{A}\,C + \overline{B}\,C$

Step 2 : Implement Boolean function with AND-OR logic.

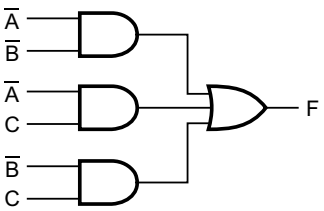


Fig. 2.4.5 (a)

Step 3 : Convert AND-OR logic to NAND-NAND logic.

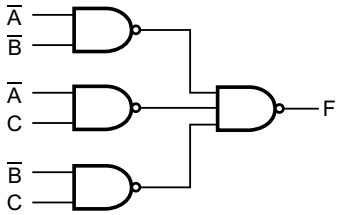


Fig. 2.4.5 (b)

Note : It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

Examples with Solutions

Example 2.4.4 *Implement EX-OR gate using only NAND gates.*

May-05, Dec.-06, 08, May-09, Marks 2

Solution : The Boolean expression for EX-OR gate is : $Y = A\overline{B} + \overline{A}B$

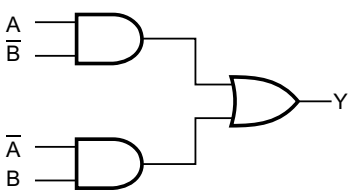


Fig. 2.4.6 (a)

We can implement AND-OR logic by using NAND-NAND logic as shown in Fig. 2.4.6 (b).

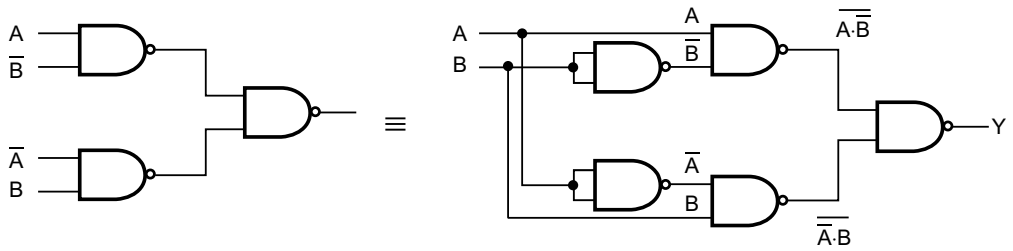


Fig. 2.4.6 (b)

Example 2.4.5 *Implement EX-NOR gate using only NAND gates.*

Solution : We have,

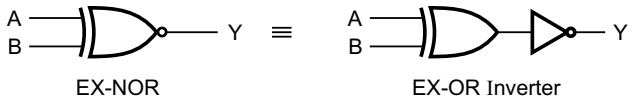


Fig. 2.4.7

Thus using result in previous example we have EX-NOR gate using only NAND gates as shown in the Fig. 2.4.7 (a).

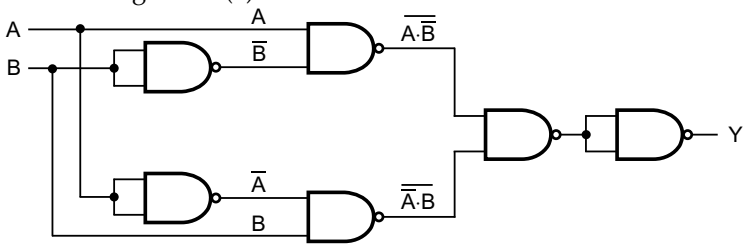


Fig. 2.4.7 (a)

Example 2.4.6 *Using K-map simplify the expression*

$Y(A,B,C,D) = m_1 + m_3 + m_5 + m_7 + m_8 + m_9 + m_0 + m_2 + m_{10} + m_{12} + m_{13}$.

Indicate the prime implicants, essential and non-essential prime implicants. Draw the logic circuit using AND-OR-INVERT gates and also using NAND gates. **May-06, Marks 16**

Solution :

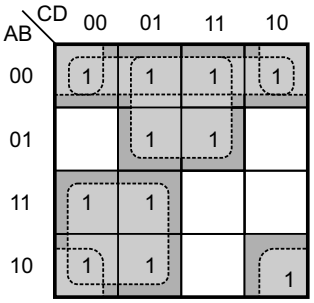


Fig. 2.4.8

Prime implicants :

$\overline{A}\,\overline{B},\ \overline{A}\,D,\ A\,\overline{C},\ \overline{B}\,\overline{D}$

Essential prime implicants :

$\overline{A}\,D,\ A\,\overline{C},\ \overline{B}\,\overline{D}$

Non-essential prime implicant : $\overline{A}\,\overline{B}$

Logic circuit :

Using AND-OR-INVERT gates

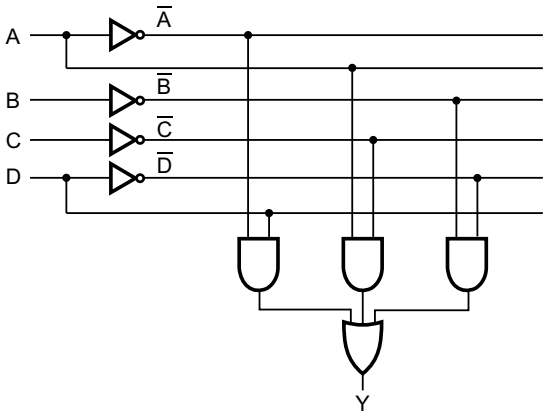


Fig. 2.4.9

Logic circuit : Using NAND gate

We can implement AND-OR logic by using NAND-NAND logic

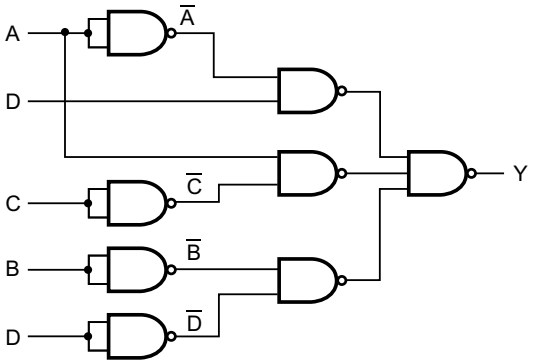


Fig. 2.4.10

Example 2.4.7 *Simplify and implement the following POS function using NAND gates.*
$f\ (A, B, C, D) = \Pi\ M\ (0, 1, 2, 3, 12, 13, 14, 15)$

Solution : Step 1 : Convert POS function into its equivalent SOP function.

$\therefore \qquad \Pi\ M\ (0, 1, 2, 3, 12, 13, 14, 15) \ = \ \sum\ m\ (4, 5, 6, 7, 8, 9, 10, 11)$

Step 2 : K-map simplification :

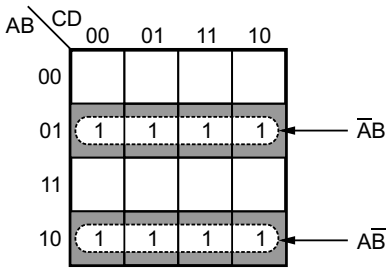
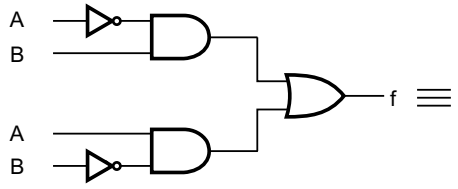


Fig. 2.4.11 (a)

$f\ (A, B, C, D) \ = \ \overline{A}\,B + A\,\overline{B}$

Step 3 : Implementation :

Using basic gates



$$f = \overline{(\overline{AB})\,(\overline{A\overline{B}})}$$

$$= \overline{(\overline{A}\,B)} + \overline{(A\,\overline{B})}$$

$$= (\overline{A}\,B) + (A\,\overline{B})$$

Using NAND gates

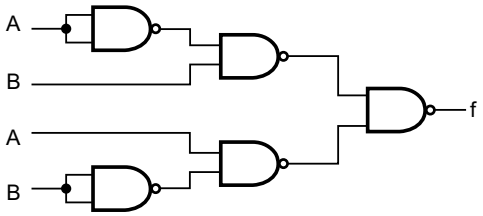


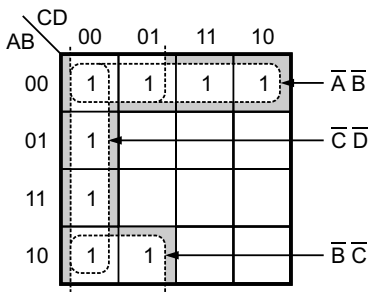
Fig. 2.4.11 (b)

Example 2.4.8 *Draw a NAND logic diagram that implements the complement of the following function.*

F (A, B, C, D) = $\sum$ *(0, 1, 2, 3, 4, 8, 9, 12)* **Dec.-10, Marks 8**

Solution :

K-map simplification



Logic diagram

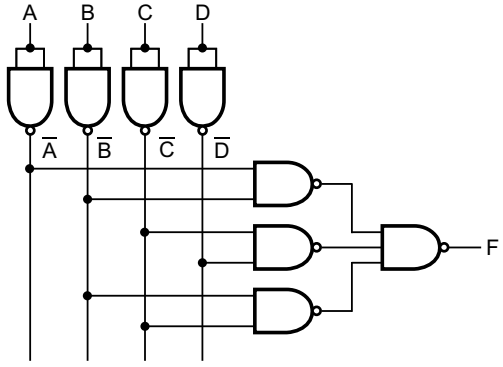


Fig. 2.4.12

Example 2.4.9 *Find a minimal sum-of-products representation for f (A, B, C, D, E) =* $\sum$ *m (1, 4, 6, 10, 20, 22, 24, 26) + d(0, 11, 16, 27) using Karnaugh map method. Draw the circuit of the minimal expression using only NAND gates.* **May-03, Marks 11**

Solution : The given function has 5 variables. Thus solving it by five variable K-map we get,

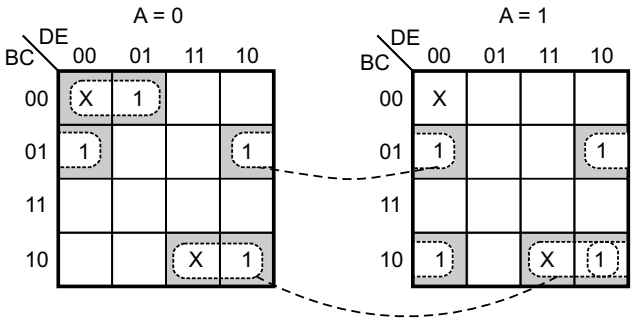


Fig. 2.4.13 (a)

$\therefore$ f(A, B, C, D, E) $= \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{B}\,C\,\overline{E} + B\,\overline{C}\,D + A\,B\,\overline{C}\,\overline{E}$

Sum of product equation can be implemented using NAND-NAND logic as shown in Fig. 2.4.13 (b).

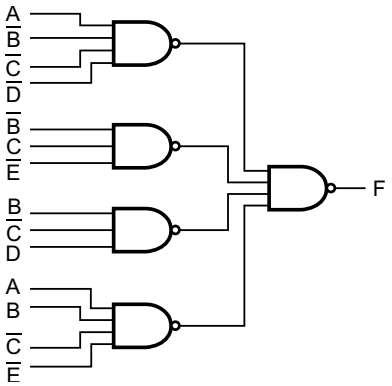


Fig. 2.4.13 (b)

Example 2.4.10 *Sketch a NAND-NAND logic circuit for the Boolean expression.*
Y = AB' + AC + BD. **Dec.-07, Marks 8**

Solution : $Y = A\,\overline{B} + AC + BD$

The SOP expression can be directly implemented by NAND-NAND logic as shown in the Fig. 2.4.14.

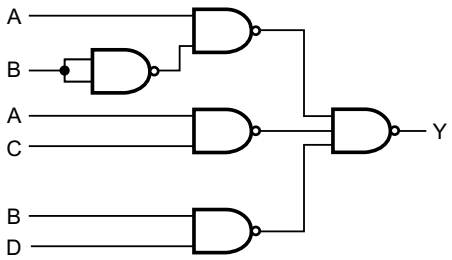


Fig. 2.4.14

Example 2.4.11 *Implement using NAND gates only, F = xyz + x'y'.* **May-12, Marks 2**

Solution :

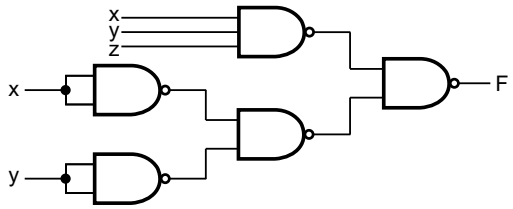


Fig. 2.4.15

Examples for Practice

Example 2.4.12 *Implement the following function with NAND gates.* $F(x,y,z)=\sum(0,6)$

Dec.-05, 12, Marks 6

Ans. :

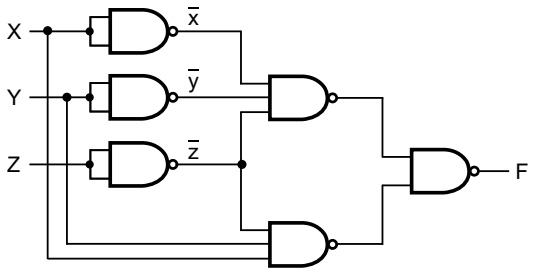


Fig. 2.4.16

Example 2.4.13 *Design a logic circuit to simulate the function* $f(A,B,C)=A(B+C)$ *by using only NAND gates.*

Dec.-04, Marks 2

Ans. :

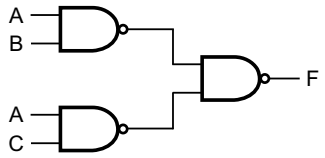


Fig. 2.4.17

2.5 NOR-NOR Implementation

Dec.-04, 05, 07, May-05, 10

The NOR function is a dual of the NAND function. For this reason, the implementation procedures and rules for NOR-NOR logic are the duals of the corresponding procedures and rules developed for NAND-NAND logic.

The implementation of a Boolean function with NOR-NOR logic requires that the function be simplified in the product of sum form. In product of sum form, we implement all sum terms using OR gates. This constitutes the first level. In the second level all sum terms are logically ANDed using AND gate. The relationship between OR-AND logic and NOR-NOR is explained using following example.

Consider the Boolean function : Y = (A + B + C) (D + E) F

This Boolean function can be implemented using OR-AND logic, as shown in the Fig. 2.5.1 (a). Fig. 2.5.1 (b) shows the OR gates are replaced by NOR gates and the AND gate is replaced by a bubbled AND gate. The implementation shown in Fig. 2.5.1 (b) is equivalent to implementation shown in Fig. 2.5.1 (a), because two bubbled on the same line represent double inversion (complementation) which is equivalent to having no bubble on the line. In case of single variable, F, the complemented variable again complemented by bubble to produce the normal value of F.

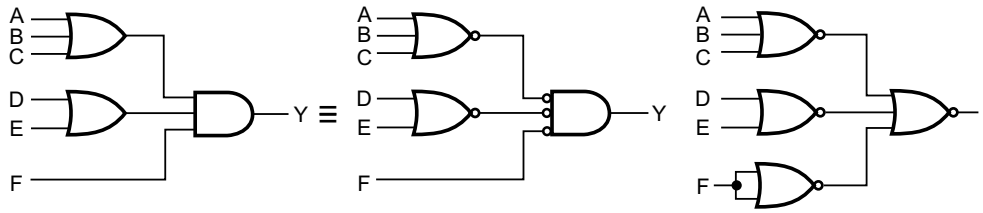


Fig. 2.5.1

In Fig. 2.5.1 (c), the output NOR gate is redrawn with the conventional symbol. The NOR gate with same inputs gives complemented result, therefore, $\overline{F}$ is replaced by NOR gate with F input to its both inputs. Thus all the three implementations of Boolean function are equivalent.

From the above example we can summarize the rules for obtaining the NOR-NOR logic diagram from a Boolean function as follows :

1. Simplify the given Boolean function and express it in product of sum form (POS form).
2. Draw a NOR gate for each sum term of the function that has two or more literals. The inputs to each NOR gate are the literals of the term. This constitute a group of first level gates.
3. If Boolean function includes any single literal or literals, draw NOR gate for each single literal and connect corresponding literal as an input to the NOR gate.
4. Draw a single NOR gate in the second level, with inputs coming from outputs of first level gates.

Illustrative Examples

Example 2.5.1 *Implement the following Boolean function with NOR-NOR logic*

$Y = A\,C + B\,C + A\,B + D.$

Solution : Step 1 : Express Boolean function in POS form.

Using duality theorem we get,

$$\overline{Y} = (\overline{A}+\overline{C})\,(\overline{B}+\overline{C})\,(\overline{A}+\overline{B})\,\overline{D}$$

Step 2 : Implement Boolean function with OR-AND logic.

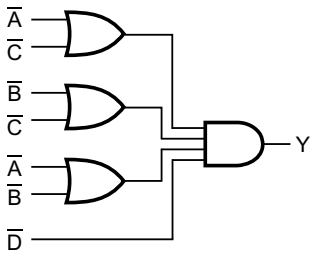


Fig. 2.5.2 (a)

Step 3 : Convert OR-AND logic to NOR-NOR logic.

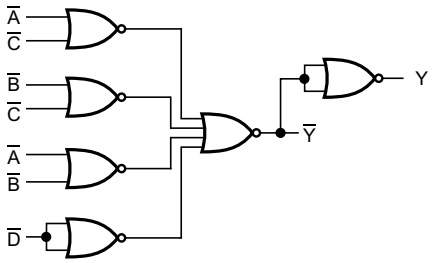


Fig. 2.5.2 (b)

Example 2.5.2 *Implement the following Boolean function with NOR-NOR logic.*

$F = (A, B, C) = \Pi\, M\,(0, 2, 4, 5, 6)$

Solution : Step 1 : Simplify the given Boolean function.

$$\therefore \quad F = (\overline{A}+B)\,C$$

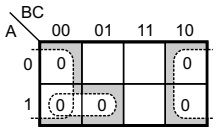


Fig. 2.5.3

Step 2 : Implement Boolean function with OR-AND logic.

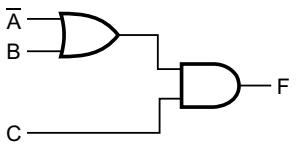


Fig. 2.5.4 (a)

Step 3 : Convert OR-AND logic to NOR-NOR logic.

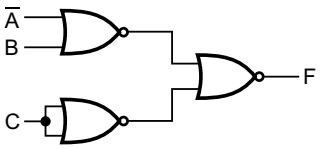


Fig. 2.5.4 (b)

Note : It is possible to directly go to step 3 skipping step 2. Here, step 2 is included for clear understanding.

Example 2.5.3 *Obtain 3-level NOR-NOR implementation of*

f (a, b, c, d, e, f) = [ab + cd] ef. **Dec.-04, Marks 2**

Solution : f (a, b, c, d, e, f) = [ab + cd] ef

$= abef + cdef = \overline{\overline{abef + cdef}}$

$= \overline{\overline{abef} \cdot \overline{cdef}} = \overline{(\bar{a}+\bar{b}+\bar{e}+\bar{f}) \cdot (\bar{c}+\bar{d}+\bar{e}+\bar{f})}$

OR-AND function can be implemented using NOR-NOR logic as shown in Fig. 2.5.5.

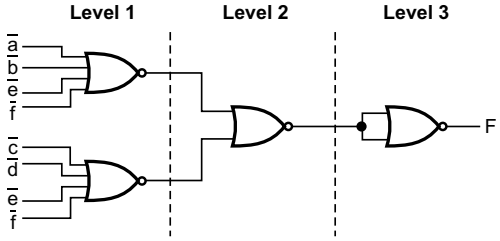


Fig. 2.5.5

Example 2.5.4 *Minimize the following using Karnaugh map. Implement the resultant function using NOR gates only.*

F (A, B, C, D, E) = Π M (2, 4, 7, 9, 26, 28, 29, 31) **May-05, Marks 16**

Solution :

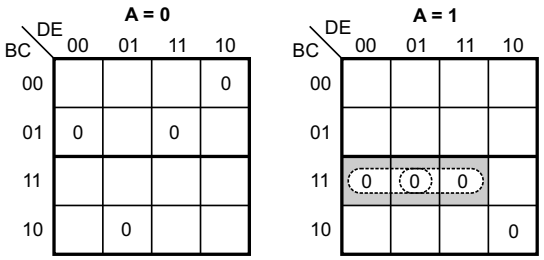


Fig. 2.5.6

F(A, B, C, D, E) = (A+B+$\overline{C}$+D+E) (A+B+$\overline{C}$+$\overline{D}$+$\overline{E}$) (A+B+C+$\overline{D}$+E) (A+$\overline{B}$+C+D+$\overline{E}$)
($\overline{A}$+$\overline{B}$+$\overline{C}$+D) ($\overline{A}$+$\overline{B}$+$\overline{C}$+$\overline{E}$) ($\overline{A}$+$\overline{B}$+C+$\overline{D}$+E)

OR-AND implementation can be replaced by NOR-NOR implementation.

Logic diagram using NOR gates

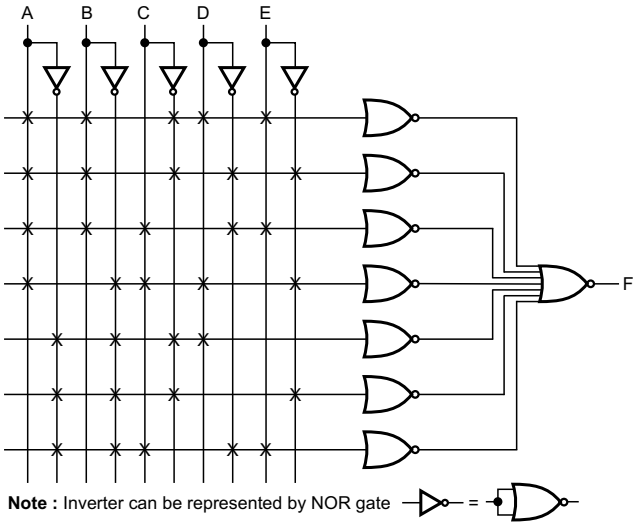


Fig. 2.5.7

Example 2.5.5 *Implement the following function using a quad 2-input NOR gates* $f = (\overline{A}B + C)\ \overline{D}$. **Dec.-05, Marks 8**

Solution :

$$f = (\overline{A}B + C)\ \overline{D}$$

$$= \overline{\overline{(\overline{A}B + C)\ \overline{D}}} = \overline{\overline{(\overline{A}B + C)} + \overline{\overline{D}}}$$

$$= \overline{\overline{(\overline{A}B + C)} + D}$$

$$= \overline{\overline{\overline{(A + \overline{B})} + C + D}} \qquad \text{where } \overline{A}\,B = \overline{\overline{A}\,B} = \overline{A + \overline{B}}$$

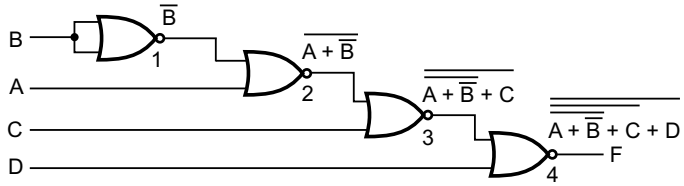


Fig. 2.5.8

Example 2.5.6 *Implement F = (AB' + A'B) (C + D') with only NOR gates.* **Dec.-07, Marks 2**

Solution :

$$F = (A\overline{B} + \overline{A}B)\ (C + \overline{D})$$

$$= A\overline{B}C + A\overline{B}\overline{D} + \overline{A}BC + \overline{A}B\overline{D}$$

$$\overline{\overline{F}} = \overline{A\overline{B}C + A\overline{B}\overline{D} + \overline{A}BC + \overline{A}B\overline{D}}$$

$$= \overline{A\overline{B}C} \cdot \overline{A\overline{B}\overline{D}} \cdot \overline{\overline{A}BC} \cdot \overline{\overline{A}B\overline{D}}$$

$$= \overline{\left(\overline{A} + B + \overline{C}\right) \cdot \left(\overline{A} + B + D\right) \cdot \left(A + \overline{B} + \overline{C}\right) \cdot \left(A + \overline{B} + D\right)}$$

This product of sum (POS) expression can be implemented by NOR-NOR logic as shown in the Fig. 2.5.9.

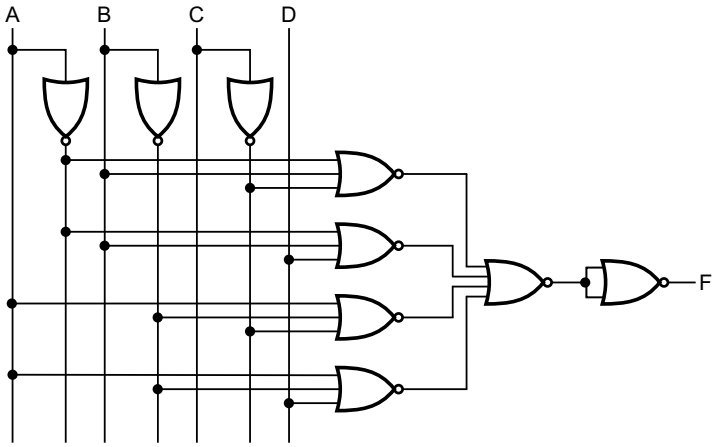


Fig. 2.5.9 Implementation using only NOR gates

Examples with Solutions

Example 2.5.7 *Implement EX-NOR gate using only NOR gates.*

Solution : The Boolean expression for EX-NOR gate is :

$$Y = AB + \overline{A}\,\overline{B} = \overline{\overline{AB} + \overline{AB}}$$

$$= \overline{\overline{AB}} \cdot \overline{\overline{AB}} = \left(\overline{A} + B\right) \cdot \left(A + \overline{B}\right)$$

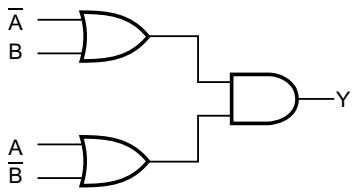


Fig. 2.5.10 (a)

We can implement OR-AND logic by using NOR-NOR logic, as shown in Fig. 2.5.10 (b).

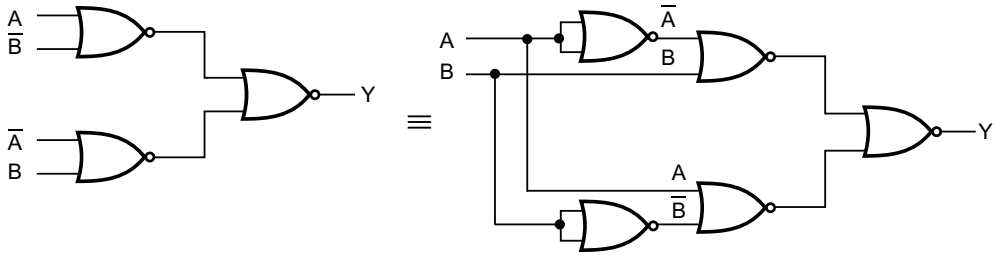


Fig. 2.5.10 (b)

Example 2.5.8 *Simplify and implement the following SOP function using NOR gates.*
$f(A, B, C, D) = \sum m\,(0, 1, 4, 5, 10, 11, 14, 15)$

Solution :

Step 1 : Convert SOP function into its equivalent POS function.

$\therefore \quad \sum m\,(0, 1, 4, 5, 10, 11, 14, 15) = \Pi M\,(2, 3, 6, 7, 8, 9, 12, 13)$

Step 2 : K-map simplification :

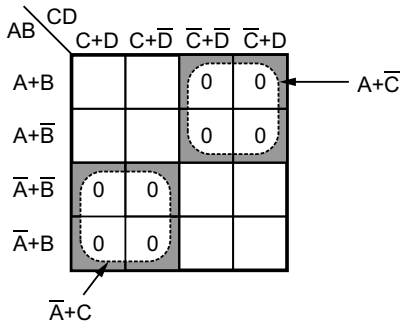
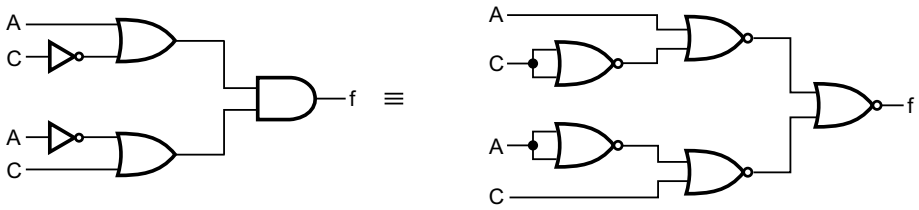


Fig. 2.5.11 (a)

$f(A, B, C, D) = (A + \overline{C})(\overline{A} + C)$

Step 3 : Implementation :

Using basic gates **Using NOR gates**



$$f = \overline{\overline{(A + \overline{C})} + \overline{(\overline{A} + C)}}$$

$$= \overline{\overline{(A + \overline{C})}}\ \overline{\overline{(\overline{A} + C)}}$$

$$= (A + \overline{C})(\overline{A} + C)$$

Fig. 2.5.11 (b)

Note : We can convert OR-AND logic into NOR-NOR logic.

Example 2.5.9 *Implement the following Boolean function with NOR-NOR logic* $F = (\overline{A} + B)C$

Solution :

Step 1 : Implement Boolean function with OR-AND logic.

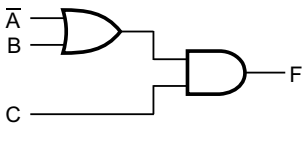


Fig. 2.5.12 (a)

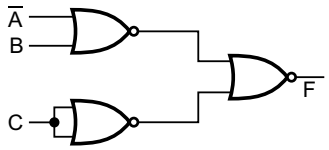


Fig. 2.5.12 (b)

Step 2 : Convert OR-AND logic to NOR-NOR logic.

Note : It is possible to directly go to step 2 skipping step 1. Here, step 1 is included for clear understanding.

Example 2.5.10 *Implement EX-OR gate using only NOR gates.*

Solution : Boolean expression of EX-OR gate $F = A\overline{B} + \overline{A}B$

$$F = \overline{\overline{F}} = \overline{\overline{A\overline{B} + \overline{A}B}}$$

$$= \overline{\overline{(A + \overline{B})} \cdot \overline{(\overline{A}B)}}$$

$$= \overline{(\overline{A} + B) \cdot (A + \overline{B})}$$

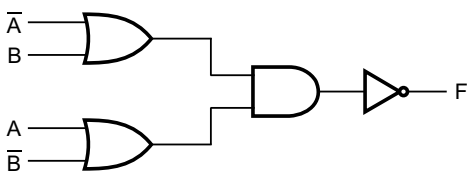


Fig. 2.5.13 (a)

Note : We can implement OR-AND logic by NOR-NOR logic.

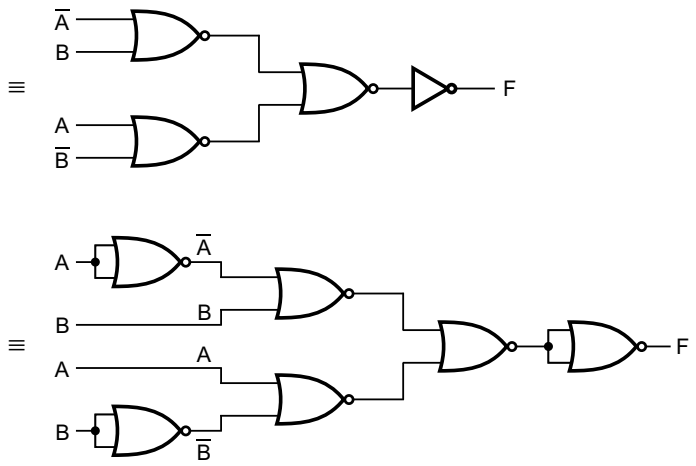


Fig. 2.5.13 (b)

Example 2.5.11 *Implement the following function using NOR gates.*
Output = 1 when the inputs are $\sum m$ *(0, 1, 2, 3, 4)*
= 0 when the inputs are $\sum m$ *(5, 6, 7).* **May-10, Marks 8**

Solution :

K-map simplification

A \ BC	B+C 00	B+$\overline{C}$ 01	$\overline{B}$+$\overline{C}$ 11	$\overline{B}$+C 10
A 0	1	1	1	1
$\overline{A}$ 1	1	0	0	0

($\overline{A}$ + $\overline{C}$) ($\overline{A}$ + $\overline{B}$)

Logic diagram

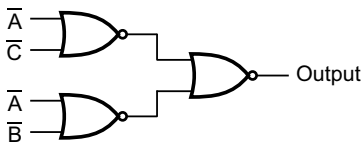


Fig. 2.5.14

2.6 Multilevel Gate Implementations

We know that logic gates can be cascaded to get the desired output. The maximum number of gates cascaded in series between a network input and the output is referred to as number of levels of gates. Thus, the Boolean functions written in sum-of-products form or in product-of-sums form are the two level gate networks. Usually, it is assumed that all variables and their complements are available as network inputs. Thus, we will not normally count inverters connected directly to input variables as a separate levels in the network. Fig. 2.6.1 shows the networks with levels of gates.

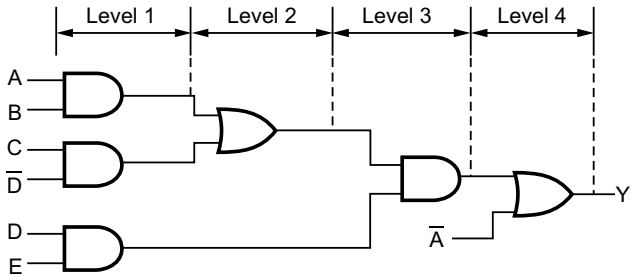


Fig. 2.6.1 (a) Four level network

Let us consider a Boolean function.

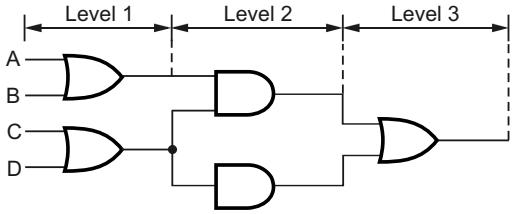


Fig. 2.6.1 (b) Three level network

$$F = \overline{A}\,\overline{C}D + \overline{A}BD + AB\overline{D} + AC\overline{D}$$

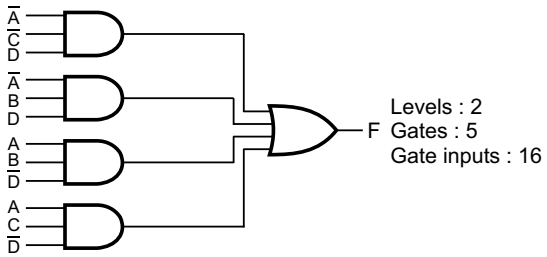


Fig. 2.6.2

This Boolean function can be implemented using two level network as shown in Fig. 2.6.2. The two level implementation requires 5 gates and 16 inputs.

Now we factorize the Boolean function and implement with gates. We observe that factorization increases the gate levels but decreases the gate inputs.

$$F = \overline{A}\,\overline{C}D + \overline{A}BD + AB\overline{D} + AC\overline{D}$$

$$= \overline{A}D(\overline{C} + B) + A\overline{D}(B + C)$$

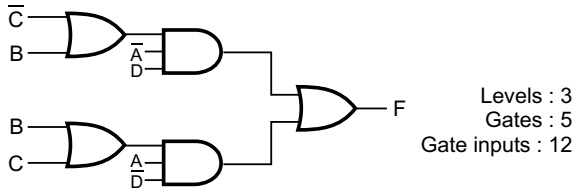


Fig. 2.6.3

Review Question

1. *Write a short note on multilevel gate implementations.*

2.7 Multilevel NAND and NOR Implementations

Dec.-11

In the previous section we have seen that how AND-OR network can be converted to NAND-NAND network and how OR-AND network can be converted to NOR-NOR network. This logic can be applied to any even level AND-OR network or OR-AND network. This is illustrated in Fig. 2.7.1 (a) and Fig. 2.7.1 (b).

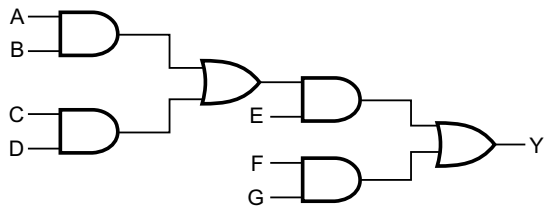


Fig. 2.7.1 (a) AND-OR network

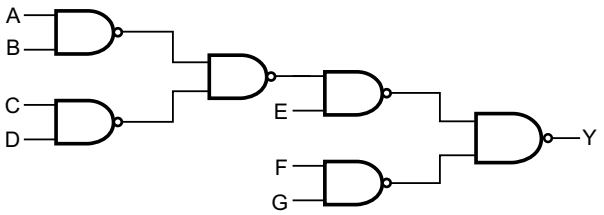


Fig. 2.7.1 (b) Equivalent NAND-NAND network

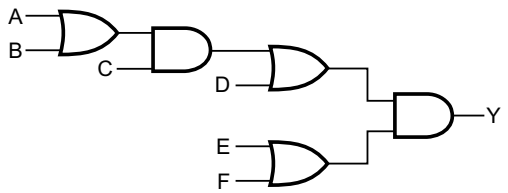


Fig. 2.7.1 (c) OR-AND network

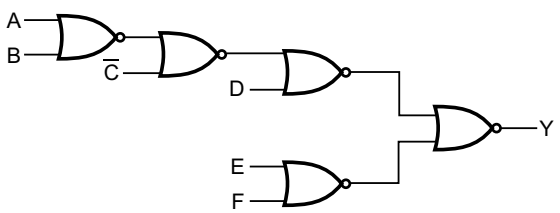


Fig. 2.7.1 (d) Equivalent NOR-NOR network

Example 2.7.1 *Draw the multiple level NAND circuit for the following expression.*
$W(X+YZ)+XY$. **Dec.-11, Marks 6**

Solution :

$$F = W(X+YZ)+XY$$

$$F = \overline{\overline{F}} = \overline{\overline{W(X+YZ)+XY}}$$

$$= \overline{\overline{W(X+YZ)} \cdot \overline{XY}}$$

$$= \overline{\overline{WX+WYZ} \cdot \overline{XY}}$$

$$= \overline{\overline{WX} \cdot \overline{WYZ} \cdot \overline{XY}}$$

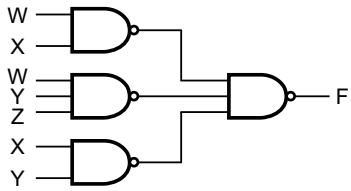


Fig. 2.7.2

Review Question

1. *Write a short note on multilevel NAND and NOR implementations.*

2.8 Multiple Output Implementations

In digital circuits, many times multiple outputs are derived from the same input variables. In such case, we can simplify each output function separately by the use of Karnaugh maps or the Quine-McCluskey technique. It is quite possible that the simplified output functions may have common terms. The common term used by one output function can be shared by other output functions. This sharing of common terms reduces the total number of gates. This is illustrated in the following example.

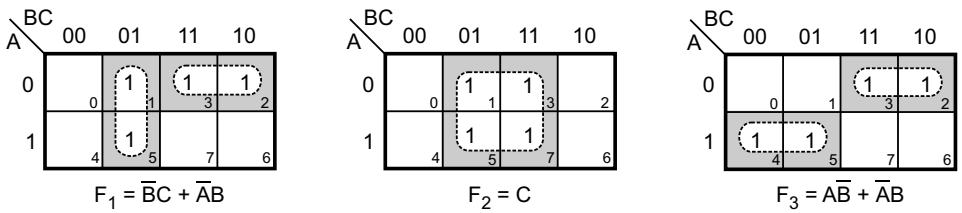
Example 2.8.1 *Simplify the following functions and draw the logic diagram for the same.*

$F_1 = f(A, B, C) = \sum (1, 2, 3, 5)$

$F_2 = f(A, B, C) = \sum (1, 3, 5, 7)$

$F_3 = f(A, B, C) = \sum (2, 3, 4, 5)$

Solution : Let us simplify the each output function by the use of Karnaugh maps, as shown in the Fig. 2.8.1.



(a) K-map for F_1 **(b) K-map for F_2** **(c) K-map for F_3**

Fig. 2.8.1 K-maps for multiple output functions F_1, F_2 and F_3

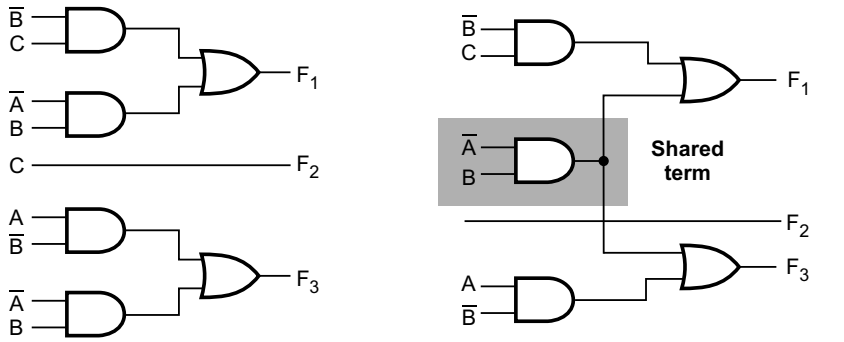
The simplified equations are :

$$F_1 = \overline{B}\,C + \overline{A}\,B$$

$$F_2 = C$$

$$F_3 = A\,\overline{B} + \overline{A}\,B$$

From above simplified equations we can observe that term $\overline{A}\,B$ is common for F_1 and F_3. This term can be shared to reduce one product term. This is illustrated in Fig. 2.8.2.



(a) Four product terms are required when each function is treated separately

(b) Three product terms are required when common term $\overline{A}\,B$ is shared

Fig. 2.8.2

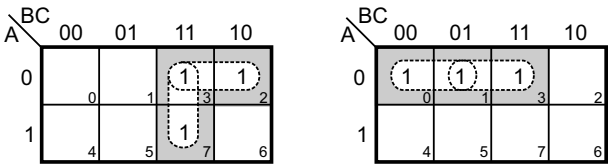
In the previous example it was very easy to identify the common term. But sometimes it is required to use special techniques to identify common terms. This is illustrated in the following example.

Example 2.8.2 *Simplify the following functions and draw the logic diagram for the same.*

$F_1 = f(A, B, C) = \sum (2, 3, 7)$

$F_2 = f(A, B, C) = \sum (0, 1, 3)$

Solution : Let us simplify the each output function by the use of Karnaugh maps, as shown in the Fig. 2.8.3.



$F_1 = BC + \overline{A}B$

(a) K-map for F_1

$F_2 = \overline{A}\,\overline{B} + \overline{A}\,C$

(b) K-map for F_2

Fig. 2.8.3 K-maps for multiple output functions F_1 and F_2

The simplified equations are :

$$F_1 = BC + \overline{A}\,B$$

$$F_2 = \overline{A}\,\overline{B} + \overline{A}\,C$$

From the above two equations we cannot identify the common term. In this case to identify the common term we have to take the help of prime implicant table.

	Prime implicants	m_0	m_1	m_2	m_3	m_7
F_1	2, 3			•	•	
	3, 7				•	•
F_2	0, 1	•	•			
	1, 3		•		•	

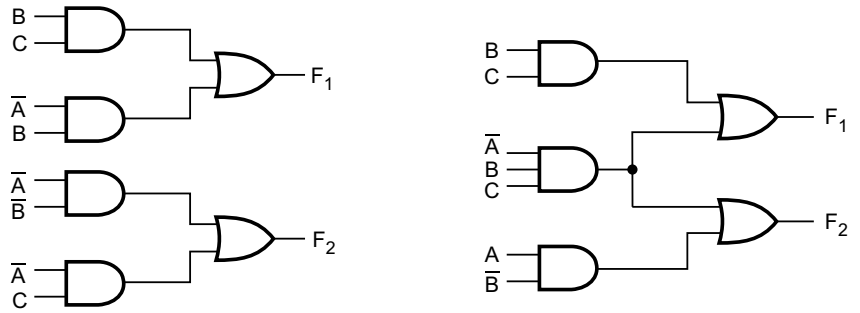
Fig. 2.8.4 Prime implicant table

From prime implicant table we can see that minterm m_3 ($\overline{A}$ B C) is common to both output functions and can therefore be shared. In the K-maps if pair corresponds to minterm m_3 is not formed then we get simplification of functions as,

$$F_1 = B\,C + \overline{A}\,B\,C$$

$$F_2 = \overline{A}\,\overline{B} + \overline{A}\,B\,C$$

The Fig. 2.8.5 shows the logic diagrams with multiple outputs considered separately and with multiple outputs considered together.



(a) Multiple outputs considered separately **(b) Multiple outputs considered together**

Fig. 2.8.5

It is important to note that the multiple outputs when considered together does not guarantee a minimum number of ICs. As seen in the previous example, multiple outputs when considered together reduced one product term, however they introduced a product term with three input variables. The reduction in the product terms in the output functions is especially helpful when we realize the circuit using PALs (Programmable Array Logics) or PLDs (Programmable Logic Devices).

Review Question

1. *Write a short note on multioutput gate implementations.*

2.9 Two Marks Questions with Answers

Q.1 ***Define binary logic?***

Ans. : Binary logic consists of binary variables and logical operations. The variables are designated by the alphabets such as A, B, C, x, y, z, etc., with each variable having only two distinct values: 1 and 0. There are three basic logic operations: AND, OR, and NOT.

Q.2 ***What is a logic gate ?***

Ans. : The logic gate is an electronic circuit that has one or more input binary variables but only one output. It is called logic gate because of it ability to operate on a number of binary inputs to perform a logical function, i.e. its output is a logical function of inputs.

Q.3 ***What are the basic digital logic gates?***

Ans. : The three basic logic gates are

- AND gate
- OR gate
- NOT gate

Q.4 ***Draw the logic symbol and construct the truth table of the following gates : a) Three input OR gate b) Three input EX-NOR gate***

Ans. : a) Three input OR gate :

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 2.9.1 Truth table for three input OR gate

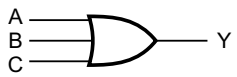


Fig. 2.9.1 Three input OR gate

b) Three input EX-NOR gate :

Inputs			Output
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Table 2.9.2 Truth table for three input EX-NOR

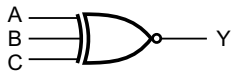


Fig. 2.9.2 Three input EX-NOR gate

Hint : For even number of ones output is one

Q.5 *Give the Boolean expression used for following gates*
a) AND b) NOR c) EX-OR d) OR e) NOT

Ans. :

a) AND : $Y = AB$

b) NOR : $Y = \overline{A + B}$

c) EX-OR : $Y = A\overline{B} + \overline{A}B = A \oplus B$

d) OR : $Y = A + B$

e) NOT : $Y = \overline{A}$

Q.6 *Which gate is equal to AND-invert Gate?*

Ans. : NAND gate.

Q.7 *Which gate is equal to OR-invert Gate?*

Ans. : NOR gate.

Q.8 *Bubbled OR gate is equal to* _____

Ans. : NAND gate

Q.9 *Bubbled AND gate is equal to* _____

Ans. : NOR gate

Q.10 *Write the names of Universal Gates*

Ans. : Universal gates are : NAND gate and NOR gate.

Q.11 *Why are NAND and NOR gates known as Universal gates ?*

Ans. : NAND and NOR are the gates that can be used alone to generate remaining gates such as NOT, AND and OR. Thus, with only any of the two gates, we can implement the logic circuit. Hence, they are called Universal gates.

Q.12 *Why digital circuits are more frequently constructed with NAND or NOR gates than with AND and OR gates ?* **Dec.-06, May-09**

Ans. : Digital circuits are frequently constructed with NAND or NOR gates due to following reasons,

- If any simple combinational circuit we want to build and we are not using universal gates then we want several AND, OR and NOT gates. This demands number of ICs, one for each type of gate. For example, IC 7408 is for 2-input AND gate, IC 7432 is for 2-input OR gate. Hence, there is possibility that all gates in each IC are not used and IC utility factor will be poor.
- By using universal gates it is possible to reduce number of ICs required to implement combinational circuit.

Q.13 *Implement EX-OR gate using only NAND gate.* **Dec.-06**

Ans. : Refer example 2.4.4.

Q.14 *Show that a bubbled AND gate works like a NOR gate.* **May-04**

Fig. 2.9.3

Ans. :

A	B	$\overline{A}$	$\overline{B}$	Y
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Here, output Y presents the output of NOR gate.

Q.15 *How can a NAND gate be used as an inverter ?* **May-04**

Ans. :

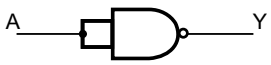


Fig. 2.9.4

Q.16 *Realize the function* $f(A, B) = \overline{A}B + A\overline{B}$ *by using only NAND gates.* **May-05; Dec.-08**

Ans. : Refer example 2.4.4.

Q.17 *Express the following switching circuit in binary logic notation.*

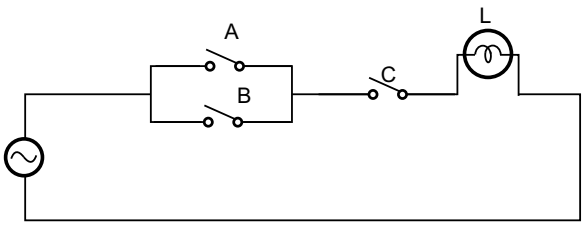


Fig. 2.9.5

Dec.-07

Ans. :

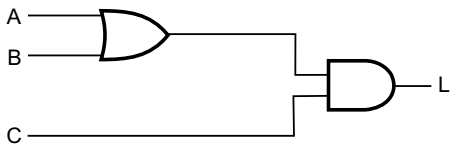


Fig. 2.9.5 (a)

Q.18 *Write down the truth table of XOR gate. Give an application for XOR function.* **May-08**

Ans. : For truth table : Refer section 2.1.

Application for XOR function : XOR function can be used as an inverter by connecting one input to logic 0. It can also be used as a modulo 2 adder.

Q.19 *What do you mean by shared term ? Explain its significance.*

Ans. : Sometimes, it may happen that simplified output functions may have common terms. The common terms which are implemented only once and used by more that one functions to generate final output are called shared terms.

Sharing of the terms reduces product terms. Sharing of common terms reduce the total number of gates.

Q.20 *Show that a positive logic NAND gate is a negative logic NOR gate.*

Dec.-09

Ans. :

Truth Table for NAND Gate		
A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

≡

Truth Table for Positive Logic NAND Gate		
A	**B**	**Y**
LOW	LOW	HIGH
LOW	HIGH	HIGH
HIGH	LOW	HIGH
HIGH	HIGH	LOW

Positive Logic

HIGH = Logic 1

LOW = Logic 0

Truth Table for NOR Gate		
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

≡

Truth Table for Negtive Logic NOR Gate		
A	**B**	**Y**
HIGH	HIGH	LOW
HIGH	LOW	HIGH
LOW	HIGH	HIGH
LOW	LOW	HIGH

Negative Logic

HIGH = Logic 0

LOW = Logic 1

Truth tables for positive logic NAND gate and negative logic NOR gates are same and hence a positive logic NAND gate is a negative logic NOR gate.

❑❑❑

3 Logic Families

Syllabus

TTL and CMOS logic and their characteristics - Tristate gates.

Contents

3.1 *Introduction*

3.2 *Characteristics of Logic Families* ***May-03, 05, 06, 07, 08, 11***
. . . . . ***Dec.-03, 04, 05, 06, 08, 10, 11*** · *Marks 2*

3.3 *Transistor Transistor Logic (TTL)* . . . . . ***Dec.-03, 04, 05, 07, 10, 12*** · · · *Marks 10*
. . . . . ***May-03, 04, 05, 06, 07, 08,***
. . . . . ***09, 10, 12*** · · · · · · · · · · · · · · · *Marks 10*

3.4 *CMOS Logic* . . . . . ***Dec.-03, 04, 06, 07, 10,*** · · · · · · *Marks 8*
. . . . . ***May-03, 04, 06, 07, 08, 11,*** · · · *Marks 12*

3.5 *Comparison between TTL and CMOS Families* . . . . . ***May-07,09, Dec.-06*** · · · · · · · · · *Marks 8*

3.6 *Interfacing TTL and CMOS Families*

3.7 *Two Marks Questions with Answers*

3.1 Introduction

A digital logic family is a group of compatible devices with the same logic levels and supply voltages. According to components used in the logic family, digital logic families are classified as shown in the Fig. 3.1.1.

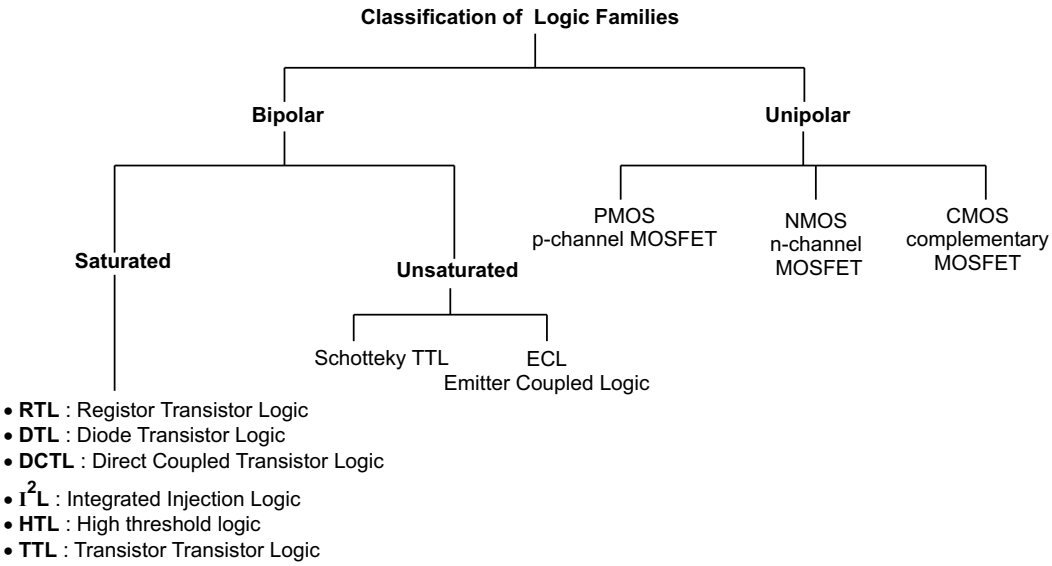


Fig. 3.1.1 Classification of logic families

Now-a-days digital integrated circuits are most commonly used in the modern digital system. They provide following advantages.

- ICs pack a lot more circuitry in a small package, so that the overall size of almost any digital system is reduced.
- The cost of ICs is very low, which makes them economical to operate.
- They have high reliability against failure, so the digital system needs less repair.
- Their reduced power consumption makes the digital system more economical to operate.
- The operating speed is higher, which makes them suitable for high-speed operations.
- The use of ICs reduces the number of external wiring connections because many of them are internal to the package.

With the widespread use of ICs, it becomes necessary to know and understand the electrical characteristics of the IC logic families such as TTL and CMOS.

Review Questions

1. *What is digital logic family ?*
2. *Give the classification of digital logic family.*

3.2 Characteristics of Logic Families

May-03, 05, 06, 07, 08, 11, Dec.-03, 04, 05, 06, 08, 10, 11

Propagation Delay

The propagation delay of a gate is basically the time interval between the application of an input pulse and the occurrence of the resulting output pulse. The propagation delay is a very important characteristic of logic circuits because it limits the speed at which they can operate. The shorter the propagation delay, the higher the speed of the circuit and vice-versa. The propagation delay is determined using two basic time intervals :

1. **t_{PLH} :** It is the delay time measured when output is changing from logic 0 to logic 1 state (LOW to HIGH)
2. **t_{PHL} :** It is the delay time measured when output is changing from logic 1 to logic 0 state (HIGH to LOW).

When t_{PHL} and t_{PLH} are not equal, the larger value is considered as a propagation delay time for that logic gate, i.e.

$$t_p = \max (t_{PLH}, t_{PHL})$$

Power Dissipation

The amount of power that an IC dissipates is determined by the average supply current, I_{CC}, that it draws from the V_{CC} supply. It is the product of I_{CC} and V_{CC}. For ICs, the value of I_{CC} for a LOW gate output (I_{CCL}) is different from a HIGH output (I_{CCH}).

Therefore, average I_{CC} is determined based on the 50 % duty cycle operation of the gate (LOW half of the time and HIGH half of the time).

$$I_{CC(avg)} = \frac{I_{CCH} + I_{CCL}}{2}$$

This can be used to calculate average power dissipation as,

$$P_{D(avg)} = I_{CC(avg)} \times V_{CC}$$

Current and Voltage Parameter

$V_{IH\ (min)}$ - High-Level Input Voltage : It is the minimum voltage level required for a logical 1 at an input. Any voltage below this level will not be accepted as a HIGH by the logic circuit.

$V_{IL\ (max)}$ - Low-Level Input Voltage : It is the maximum voltage level required for a logic 0 at an input. Any voltage above this level will not be accepted as a LOW by the logic circuit.

$V_{OH\ (min)}$ - High-Level Output Voltage : It is the minimum voltage level at a logic circuit output in the logical 1 state under defined load conditions.

$V_{OL\ (max)}$ - Low-Level Output Voltage : It is the maximum voltage level at a logic circuit output in the logical 0 state under defined load conditions.

I_{IH} - High-Level Input Current : It is the current that flows into an input when a specified high-level voltage is applied to that input.

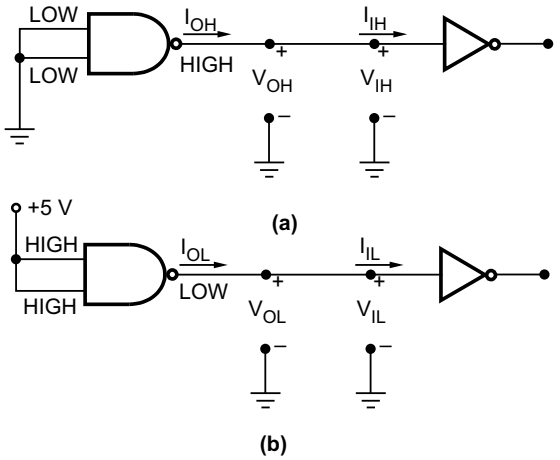


Fig. 3.2.1 Currents and voltages in the two logic states

I_{IL} - Low-Level Input Current : It is the current that flows into an input when a specified low-level voltage is applied to that input.

I_{OH} - High-Level Output Current : It is the current that flows from an output in the logical 1 state under specified load conditions.

I_{OL} - Low-Level Output Current : It is the current that flows from an output in the logical 0 state under specified load conditions.

Note : The current directions shown in the Fig. 3.2.1 may be opposite to those shown depending on the logic family.

Noise Margin and Logic Voltages Levels

In digital circuits, the binary 0 and 1 are represented by a pair of voltage levels. Each logic family has a different standard. The Table 3.2.1 shows the voltages used by several families.

Family	Logic 0	Logic 1
TTL	0 V	+ 5 V
ECL	– 1.7 V	– 0.9 V
CMOS	0 V	3 - 15 V

Table 3.2.1

These are the ideal voltage levels. But in practice, it is difficult to get these ideal voltages. Stray electric and magnetic field can induce voltages on the connecting wires between logic circuits. These unwanted signals are called noise and can sometimes cause the voltage at the input to a logic circuit to drop below $V_{IH(min)}$ or rise above $V_{IL(max)}$, which could produce unpredictable operation. The noise immunity of at a logic circuit refers to the circuit's ability to tolerate the noise without causing spurious changes in the output voltage. To avoid this problem due to noise, voltage level $V_{IH(min)}$ is kept at a few fraction of volts below $V_{OH(min)}$ and voltage level $V_{IL(max)}$ is kept above $V_{OL(max)}$, at the design time. This is illustrated in Fig. 3.2.2.

As shown in Fig. 3.2.2, V_{NH} is the difference between the lowest possible HIGH output, $V_{OH(min)}$ and the minimum voltage, $V_{IH(min)}$ required for a HIGH input. This voltage difference, V_{NH} is called high-state noise margin. Similarly, we have low-state noise margin. It is the voltage difference between the largest possible low output, $V_{OL(max)}$ and the maximum voltage, $V_{IL(max)}$ required for a LOW input.

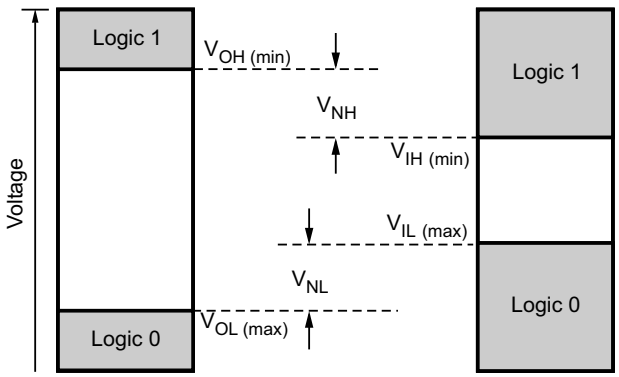


Fig. 3.2.2 Noise margins

In short we can write as,

$$V_{NH} = V_{OH(min)} - V_{IH(min)} \text{ and}$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

- The noise margin allows the digital circuit to function properly if noise voltages are within the limits of V_{NH} and V_{NL} for a particular logic family.

Fan-in and Fan-out

- In a digital system, we typically find many types of digital ICs interconnected to perform various functions. In these situations, the output of a logic gate may be connected to the inputs of several other similar gates. The maximum number of inputs of several gates that can be driven by the output of a logic gate is decided by the parameter called **fan-out**. In general, the fan-out is defined as the maximum number of inputs of the same IC family that the gate can drive maintaining its output levels within the specified limits. For example, a logic gate with fan-out 10 can drive maximum 10 logic inputs from the same family. It depends on current sourcing and sinking capacity of input and output signals of same IC family.
- The **fan-in** of a digital logic gate refers to the number of inputs. For example, an inverter has a fan-in of 1, a 2-input NOR gate has a fan-in of 2, a 4-input NAND gate has a fan-in of 4 and so on. A logic designer has to select the fan-in of the gate to accommodate the number of inputs. At the hardware level, however, the fan-in provides information about the intrinsic speed of the gate itself. In general, the propagation delay increases with the fan-in. This means that 2-input NAND gate is faster than the 4-input NAND if both are from same logic family.

Current Sinking

- A device output is said to **sink current** when current flows from the power supply, through the load and through the device output to ground. This is illustrated in Fig. 3.2.3 (a).

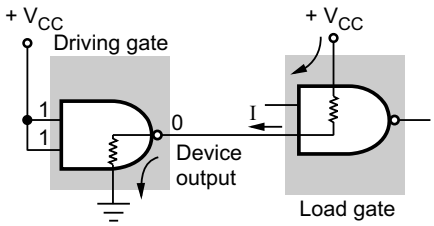


Fig. 3.2.3 (a) Current sinking

Current Sourcing

- A device output is said to **source current** when current flows from the power supply, out of the device output and through the load to ground. This is illustrated in Fig. 3.2.3 (b).

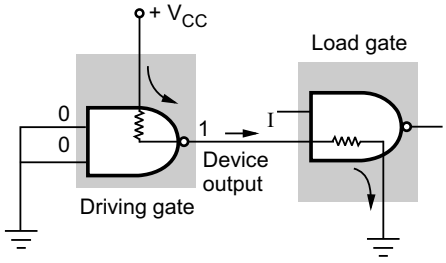


Fig. 3.2.3 (b) Current sourcing

Speed Power Product (Figure of Merit)

- In general, for any digital IC, it is desirable to have shorter propagation delays (higher speed) and lower values of power dissipation. There is usually a trade-off between switching speed and power dissipation in the design of a logic circuit i.e. speed is gained at the expense of increased power dissipation. Therefore, a common means for measuring and comparing the overall performance of an IC family is the **Speed-Power Product (SPP)**. **It is also called Figure of Merit.**

Illustrative Examples

Example 3.2.1 *If a manufacture specifies the minimum logical 1 at a gate output a 4.0 V and also specifies that any voltage done upto 3.6 V will be considered as logical 1, find the noise margin.* **May-03, Marks 2**

Solution : Given : $V_{OH(min)} = 4.0\ V$ and $V_{IH(min)} = 3.6\ V$

$\therefore$ Noise Margin $= V_{OH(min)} - V_{IH(min)} = 4.0\ V - 3.6\ V = 0.4\ V$

Example 3.2.2 *Determine the fan-out given $I_{IH(max)} = 40\ \mu A$ and $I_{OH(max)} = 400\ \mu A$.* **May-04, Marks 2**

Solution : $$\text{fan-out} = \frac{I_{OH(max)}}{I_{IH(max)}} = \frac{400\mu A}{40\mu A} = 10$$

Example 3.2.3 *A certain gate draws 2 mA when its output is high and 3.6 mA when its output is low. What is the average power dissipation if V_{CC} is 5 V and it is operated on a 50 % duty cycle ?* **Dec.-06, Marks 2**

Solution : Given $I_{OH} = 2\ mA$

$I_{OL} = 3.6\ mA$

$$P_D = \frac{2\ mA + 3.6\ mA}{2} \times 5 = 2.8\ mA \times 5 = 14\ mW$$

Example 3.2.4 *For a certain IC family, propagation delay is 10 ns with an average power dissipation of 6 mW. What is its speed power product ?*

Solution : The speed-power product is given as,

SPP = Propagation delay × Average power dissipation

= 10 ns × 6 mW = 60 pico joules (pJ)

Example 3.2.5 *If $V_{CC} = 5\ V$, $I_{CC(0)} = 22\ mA$, $I_{CC(1)} = 8\ mA$, $V_{OH} = 2.4\ V$, $V_{IH} = 2\ V$, $V_{OL} = 0.45\ V$, $V_{IL} = 0.8\ V$, $t_{PHL} = 15\ nsec$, $t_{PLH} = 22\ nsec$, $I_{IH} = 40\ \mu A$, $I_{OH} = 5.2\ mA$, calculate :*

i) Noise margin logic 1 and logic 0.

ii) Power dissipation.

iii) Propagation delay and figure of merit.

iv) Fan-out.

Solution :

i) Noise margin logic 1

$$V_{NH} = V_{OH(min)} - V_{IH(min)} = 2.4\ V - 2\ V$$

$$V_{NH} = 0.4\ V$$

Noise margin logic 0

$$V_{NL} = V_{IL(max)} - V_{OL(max)} = 0.8\ V - 0.45\ V$$

$$V_{NL} = 0.35\ V$$

ii) Power dissipation = $I_{CC(avg)} \times V_{CC}$

$$\therefore \quad I_{CC(avg)} = \frac{I_{CC(0)} + I_{CC(1)}}{2} = 15\ mA.$$

$$\therefore \quad \text{Power dissipation} = 15\ mA \times 5\ V = 75\ mW.$$

iii) Propagation delay

$$t_p = \max(t_{PLH} \cdot t_{PHL})$$

$$\therefore \quad t_p = t_{PLH} = 22\ nsec$$

$$\text{OR} \quad t_p = \frac{1}{2}(t_{PLH} + t_{PHL}) = \frac{1}{2}(22+15)\ nsec = 18.5\ nsec$$

Figure of merit = SPP = Delay × Power dissipation = 18.5 nsec × 75 mwat

= 1387.5 pico joule

iv) Fan-out $= \frac{|I_{OH(max)}|}{I_{IH(max)}} = \frac{5.2\ mA}{40\ \mu A} = 130$

Review Questions

1. *Define fan-in and fan-out.* **May-03, 06, 11, Dec.-03, 05, 08, 10, 11, Marks 2**
2. *Differentiate : Source and sink current.* **May-07, Marks 2**
3. *Define power dissipation and propagation delay.* **May-03, 05, 06, Marks 2**
4. *Why does the propagation delay occur in logic circuits ?* **Dec.-06, Marks 2**
5. *Define noise margin.* **May-05, 07, 08, Dec.-03, 04, 08, 10, Marks 2**

3.3 Transistor Transistor Logic (TTL)

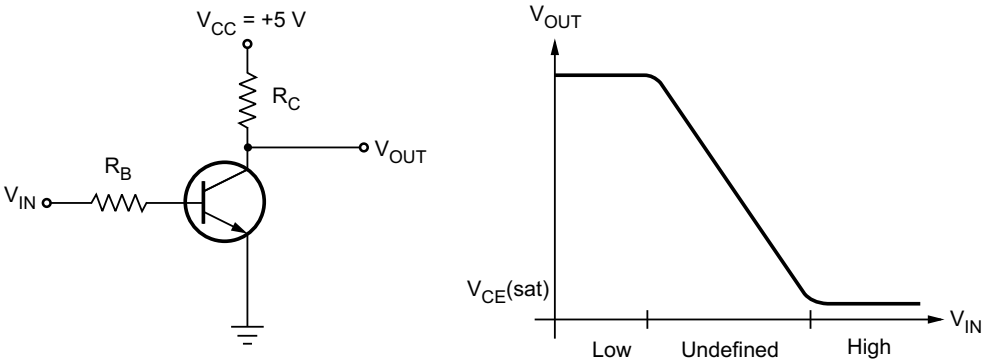
May-03, 04, 05, 06, 07, 08, 09, 10, 12, Dec.-03, 04, 05, 07, 10, 12

Transistor transistor logic, TTL, is named for its dependence on transistors alone to perform basic logic operations. The first version, which is now known as standard TTL,

was developed in 1965 and is rarely used in today's systems. Through the years, the basic design has been modified to improve its performance in several respects and as a consequence, a number of subfamilies have evolved. In this section we are going to study the basic transistor configurations in TTL and its subfamily circuits along with their characteristics.

3.3.1 TTL Inverter

We have seen that when the input voltage is low, the output voltage is HIGH and vice versa. Therefore, we can make a logic inverter from an npn transistor in the common emitter configuration as shown in the Fig. 3.3.1.



(a) Circuit diagram **(b) Transfer characteristics**

Fig. 3.3.1

The Fig. 3.3.2 (a) and 3.3.2 (b) shows the operation of transistor inverter for both the inputs (HIGH and LOW) using switching analogy.

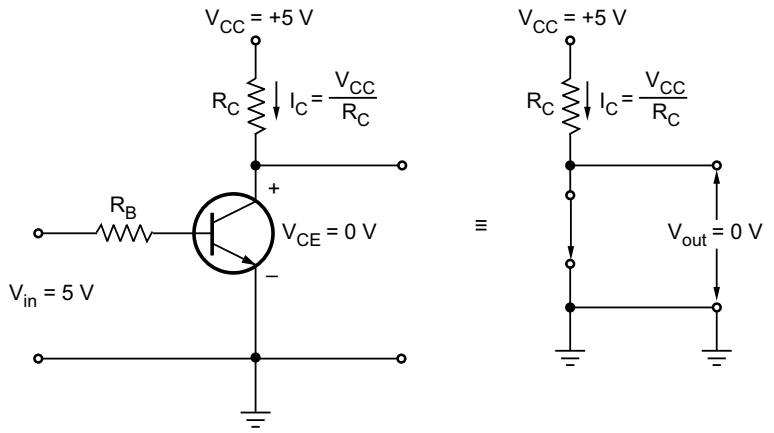


Fig. 3.3.2 (a)

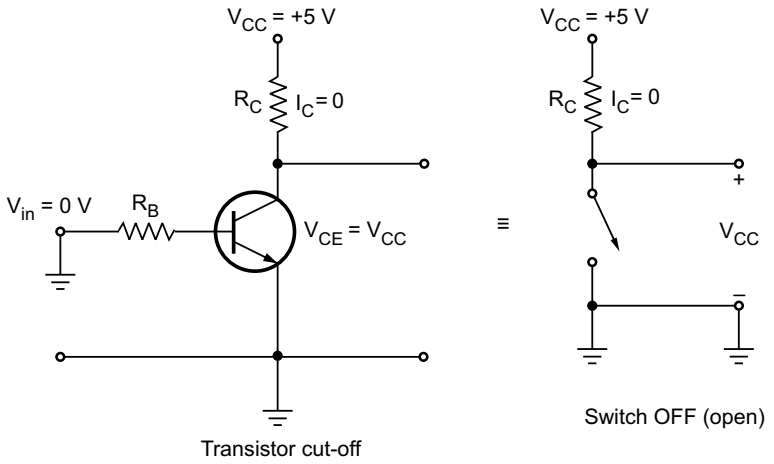


Fig. 3.3.2 (b)

3.3.2 2-Input TTL NAND Gates

The Fig. 3.3.3 (a) shows the circuit diagram of 2-input NAND gate. Its input structure consists of multiple-emitter transistor and output structure consists of totem-pole output. Here, Q_1 is an NPN transistor having two emitters, one for each input to the gate. Although this circuit looks complex, we can simplify its analysis by using the diode equivalent of the multiple-emitter transistor Q_1, as shown in Fig. 3.3.3 (b). Diodes D_2 and D_3 represent the two E-B junctions of Q_1 and D_4 is the collector-base (C-B) junction.

The input voltages A and B are either LOW (ideally grounded) or HIGH (ideally + 5 volts). If either A or B or both are low, the corresponding diode conducts and the base of Q_1 is pulled down to approximately 0.7 V. This reduces the base voltage of Q_2 to almost zero. Therefore, Q_2 cuts off. With Q_2 open , Q_4 goes into cut-off and the Q_3 base is pulled HIGH. Since Q_3 acts as an emitter follower, the Y output is pulled up to a HIGH voltage. On the other hand, when A and B both are HIGH, the emitter diode of Q_1 are reversed biased making them off. This causes the collector diode D_4 to go into forward

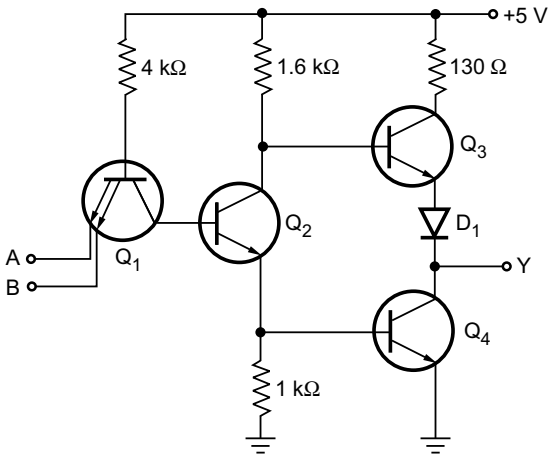


Fig. 3.3.3 (a) Two input TTL NAND gate

conduction. This forces Q_2 base to go HIGH. In turn, Q_4 goes into saturation, producing a low output. Table 3.3.1 summarizes all input and output conditions.

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Table 3.3.1 Truth table for 2-input NAND gate

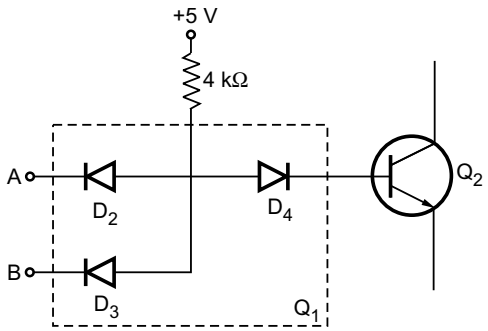


Fig. 3.3.3 (b) Diode equivalent for Q_1

Without diode D_1 in the circuit, Q_3 will conduct slightly when the output is low. To prevent this, the diode is inserted; its voltage drop keeps the base-emitter diode of Q_3 reverse-biased. In this way, only Q_4 conducts when the output is low.

3.3.3 3-Input TTL NAND Gate

The Fig. 3.3.4 shows the three input TTL NAND gate. It is same as that of two input TTL NAND gate except that its Q_1(NPN) transistor has three emitters instead of two. Rest of the circuit is same. For three input NAND gate if all the inputs are logic 1 then and then only output is logic 0; otherwise output is logic 1. The operation is similar to the 2-input NAND gate. The Table 3.3.2 shows the truth table for 3-input NAND gate.

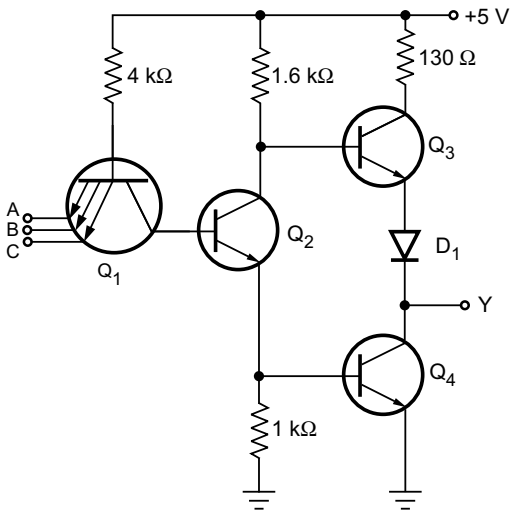


Fig. 3.3.4 Three input TTL NAND gate

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 3.3.2 Truth table of 3-input NAND gate.

3.3.4 Totem-Pole Output

Fig. 3.3.5 shows an highlighted output configuration. Transistor Q_3 and Q_4 form a totem-pole. Such a configuration is known as **active pull-up** or **totem** pole output. The active pull-up formed by Q_3 and Q_4 has specific advantage. Totem-pole transistors are used because they produce a LOW output impedance. Either Q_3 acts as an emitter follower (HIGH output) or Q_4 is saturated (LOW output). When Q_3 is conducting, the output impedance is approximately 70 Ω ; when Q_4 is saturated, the output impedance is only 12 Ω . Either way, the output impedance is low. This means that the output voltage can change quickly from one state to the other because any stray output capacitance is rapidly charged or discharged through the low output impedance. Thus the propagation delay is low in totem-pole TTL logic.

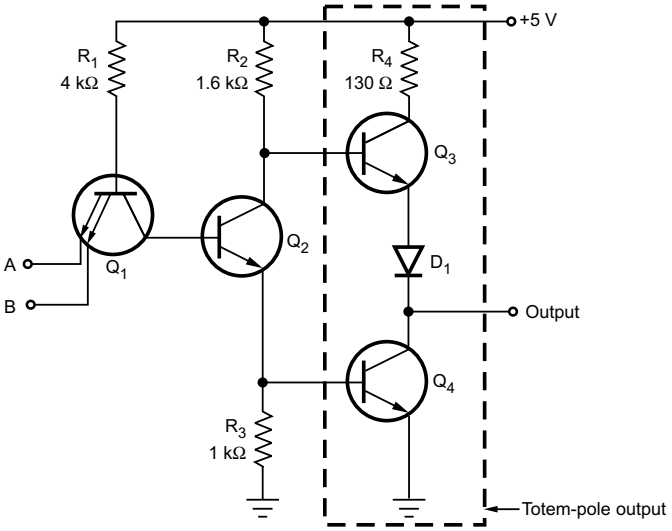


Fig. 3.3.5 Two input NAND gate with totem-pole output

3.3.5 Open-Collector Output

One problem with totem pole output is that two outputs cannot be tied together. See the Fig. 3.3.6, where the totem pole outputs of two separate gates are connected together at point X. Suppose that the output of gate A is high (Q_{3A} ON and Q_{4A} OFF) and the output of gate B is low (Q_{3B} OFF and Q_{4B} ON). In this situation transistor Q_{4B} acts as a load for Q_{3A}. Since Q_{4B} is a low resistance load, it draws high current around 55 mA. This current might not damage Q_{3A} or Q_{4B} immediately, but over a period of time can cause overheating and deterioration in performance and eventual device failure. (See Fig. 3.3.6 on next page)

Some TTL devices provide another type of output called **open collector output**. The outputs of two different gates with open collector output can be tied together. This is known as **wired logic**. Fig. 3.3.7 shows a 2-input NAND gate with an open-collector output eliminates the pull-up transistor Q_3, D_1 and R_4. The output is taken from the open collector terminal of transistor Q_4.

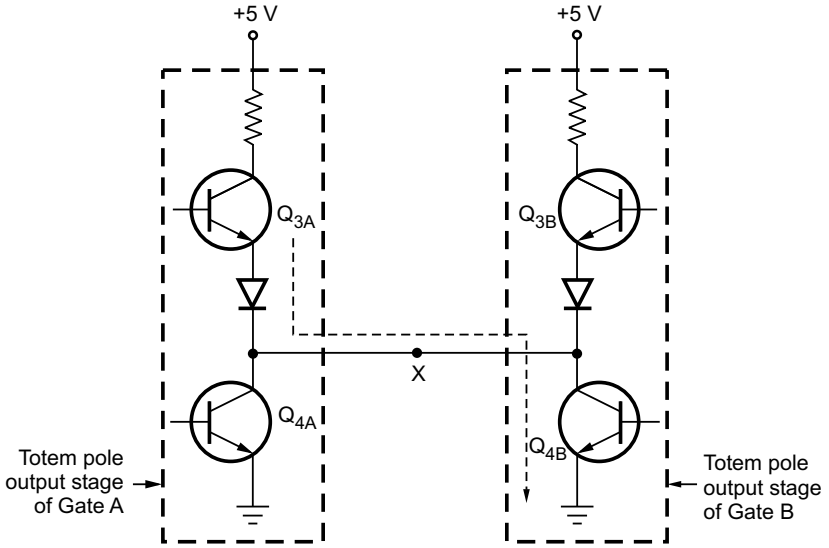


Fig. 3.3.6 Totem-pole outputs tied together can produce harmful current

Because the collector of Q_4 is open, a gate like this will not work properly until you connect an external pull-up resistor, as shown in Fig. 3.3.8. When Q_4 is ON, output is low and when Q_4 is OFF output is tied to V_{CC} through an external pull up resistor.

As mentioned earlier, the open collector outputs of two or more gates can be connected together, as shown in the Fig. 3.3.9 (a). The connection is called a wired-AND and represented schematically by the special AND gate symbol as shown in Fig. 3.3.9 (b).

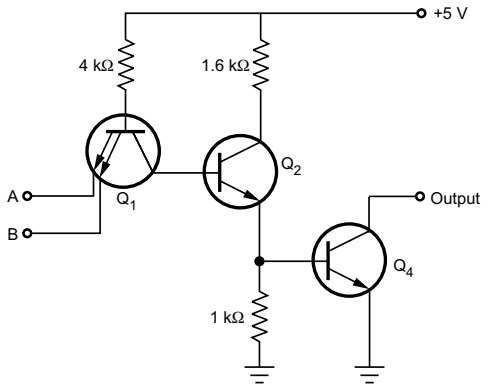


Fig. 3.3.7 Open collector 2-input TTL NAND gate

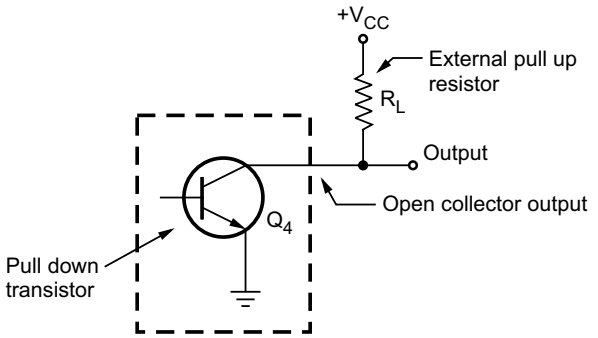
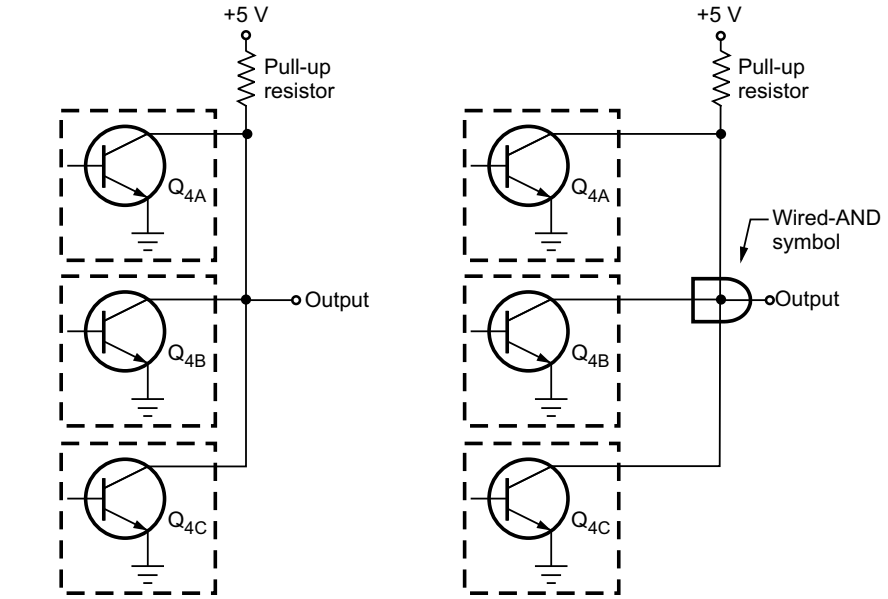


Fig. 3.3.8 Open collector output with pull-up resistor



(a) Open collector outputs connected together

(b) Wired-AND output with special AND gate symbol

Fig. 3.3.9

Example 3.3.1 *A two input NAND gate has V_{CC} = +5 V and 1 kΩ load connected to its output. Calculate the output voltage,*
(a) When both input are LOW (b) When both input are HIGH.

Solution : a) When both inputs of NAND gate are LOW, the output is HIGH and it is given as,

$$V_{OH} = V_{CC} - V_{CE(sat)} - V_D - I_L \times (130\ \Omega) = 5 - 0.1 - 0.7 - I_L\ (130\ \Omega)$$

$$= 4.2\ V - I_L\ (130\ \Omega) \quad \ldots (1)$$

where the load current is,

$$I_L = \frac{V_{OH}}{R_L} = \frac{V_{OH}}{1\ k\Omega}$$

Substituting for I_L in the equation 1 we get,

$$V_{OH} = 4.2\ V - \frac{V_{OH}}{1\ k\Omega}\ (130\ \Omega)$$

$$\therefore \quad V_{OH} + \frac{130\ V_{OH}}{1000} = 4.2$$

$$\therefore \quad 1130\ V_{OH} = 4200$$

$$\therefore \quad V_{OH} = \frac{4200}{1130} = 3.716\ V$$

b) When both inputs of NAND gate are HIGH, the output is LOW and it is given as,

$$V_{OL} = V_{CE(sat)} \approx 0.1\ V$$

3.3.6 Comparison between Totem-Pole and Open-Collector Outputs

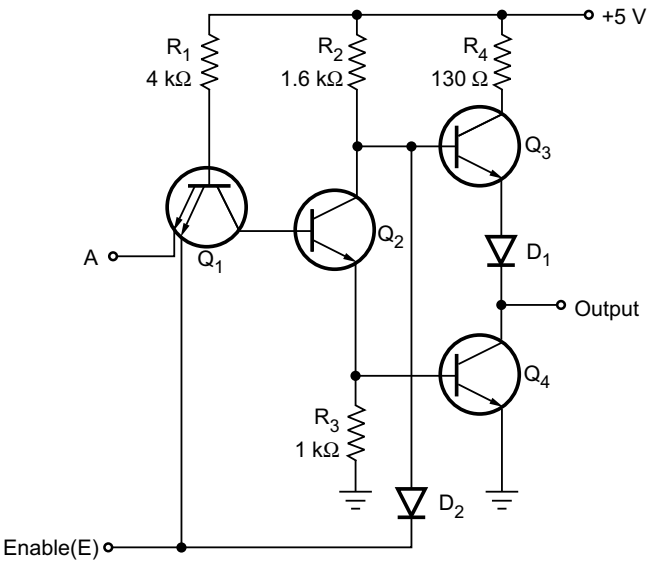
Table 3.3.3 summarizes the difference between totem-pole and open collector outputs.

Sr. No.	Totem-pole	Open collector
1.	Output stage consists of pull-up transistor (Q_3), diode resistor and pull-down transistor (Q_4).	Output stage consists of only pull-down transistor.
2.	External pull-up resistor is not required.	External pull-up resistor is required for proper operation of gate.
3.	Output of two gates cannot be tied together.	Output of two gates can be tied together using wired AND technique.
4.	Operating speed is high.	Operating speed is low.

Table 3.3.3 Comparison of totem-pole and open collector output

3.3.7 Tri-State TTL Inverter

The tristate configuration is a third type of TTL output configuration. It utilizes the high-speed operation of the totem-pole arrangement while permitting outputs to be wired-ANDed (connected together). It is called tristate TTL because it allows three possible output stages : HIGH, LOW and high-impedance. We know that transistor Q_3 is ON when output is HIGH and Q_4 is ON when output is LOW. In the high impedance state both transistors, transistors Q_3 and Q_4 in the totem-pole arrangement are turned OFF. As a result, the output is open or floating, it is neither LOW nor HIGH.



A	E	Output
0	1	1
1	1	0
X	0	Floating

Truth table

Fig. 3.3.10 Tristate TTL inverter

Fig. 3.3.10 shows the simplified circuit for tristate inverter. It has two inputs A and E. A is the normal logic input whereas E is an ENABLE input. When ENABLE input is HIGH, the circuit works as a normal inverter. Because when E is HIGH, the state of the transistor Q_1 (either ON or OFF) depends on the logic input A, and the additional component diode is open circuited as its cathode is at logic HIGH. When ENABLE input is LOW, regardless of the state of logic input A, the base-emitter junction of Q_1 is forward biased and as a result it turns ON. This shunts the current through R_1 away from Q_2 making it OFF. As Q_2 is OFF, there is no sufficient drive for Q_4 to conduct and hence Q_4 turns off. The LOW at ENABLE input also forward-biases diode D_2, which shunt the current away from the base of Q_3, making it OFF. In this way, when ENABLE input is LOW, both transistors are OFF and output is at high impedance state.

2-input tri-state NAND gate

If we add one more emitter to the transistor Q_1 of the tri-state inverter circuit of Fig. 3.3.10 as a second input we get the circuit of 2-input tri-state NAND gate. The circuit will work as a normal 2-input NAND gate when ENABLE input is HIGH. When ENABLE input is low, both output transistors Q_3 and Q_4 are OFF to drive output in tri-state. The Table 3.3.4 shows the truth table for 2-input tri-state NAND gate.

A	B	E	Output
0	0	1	1
0	1	1	1
1	0	1	1
1	1	1	0

Table 3.3.4 Truth table for tri-state 2-input NAND gate

Fig. 3.3.11 shows the logic symbols for tristate inverter. In above case circuit operation is enabled when ENABLE input is HIGH. Therefore, ENABLE input is active high. The logic symbol for active high enable input is shown in Fig. 3.3.11 (a). In some circuits ENABLE input can be active LOW, i.e. circuit operates when ENABLE input is LOW. The logic symbol for active low ENABLE input is shown in the Fig. 3.3.11 (b).

(a) Logic symbol for active high enable input **(b) Logic symbol for active low enable input**

Fig. 3.3.11

As mentioned earlier, tristate outputs can be connected together. This is illustrated in Fig. 3.3.12. When CONTROL input is LOW,

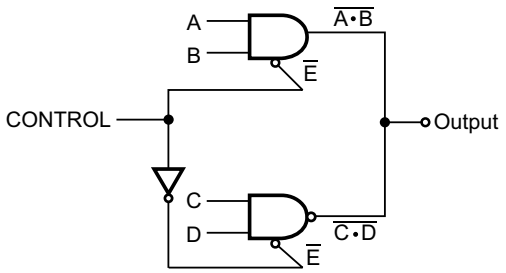


Fig. 3.3.12 Tristate outputs connected together

upper NAND gate is enabled and its output ($\overline{A \cdot B}$) is available on the output terminal. The output of lower NAND gate is in the tristate. When CONTROL input is high, the output of upper gate is in the tristate while output of lower NAND gate ($\overline{C \cdot D}$) is available on the output terminal. The same concept is used when two or more device shares a common bus.

3.3.8 Characteristics of TTL Family

In 1964 Texas Instruments Corporation introduced the standard TTL ICs, 54/74 series. There are several series/ subfamilies in the TTL family of logic devices. Let us see the characteristics of standard TTL family.

Supply voltage and temperature range

Both the 74 series and 54 series operate on supply voltage of 5 V. The 74 series works reliably over the range 4.75 V to 5.25 V, while the 54 series can tolerate a supply variation of 4.5 to 5.5 V. The 74 series devices are guaranteed to work reliably over a temperature range of 0 to 70 °C where as 54 series devices can handle temperature variations from – 55 to +125 °C. From the above values we can say that 54 series devices have greater tolerance of voltage and temperature variations. Hence, these devices are used where it is necessary to maintain reliable operation over an extreme range of conditions. For example, in military and space application. The only disadvantage of these devices is that they are expensive.

Voltage levels and noise margin

Table 3.3.5 shows the input and output logic voltage levels for the standard 74 series. The minimum and maximum values shown in the Table 3.3.5 are for worst case conditions of power supply, temperature and loading conditions.

Looking at the Table 3.3.5 we can say that, in the worst case, there is difference of 0.4 V between the driver output voltages and the required load input voltages. For instance, the worst-case low values are

$V_{OL(max)}$ = 0.4 V driver output

$V_{IL(max)}$ = 0.8 V load input

Similarly, the worst-case high values are

$V_{OH(min)}$ = 2.4 V driver output

$V_{IH(min)}$ = 2 V load input

Voltages	Minimum	Typical	Maximum
V_{OL}	–	0.2	0.4
V_{OH}	2.4	3.4	–
V_{IL}	–	–	0.8
V_{IH}	2.0	–	–

Table 3.3.5 Voltage levels

In either case, the difference is 0.4 V. This difference is called *noise margin*. For TTL, Low state noise margin, V_{NL} and high state noise margin, V_{NH} both are equal and 0.4 V. This is illustrated in Fig. 3.3.13. It provides built-in protection against noise. It ensures reliable operation of the device for induced noise voltages less than 0.4 V.

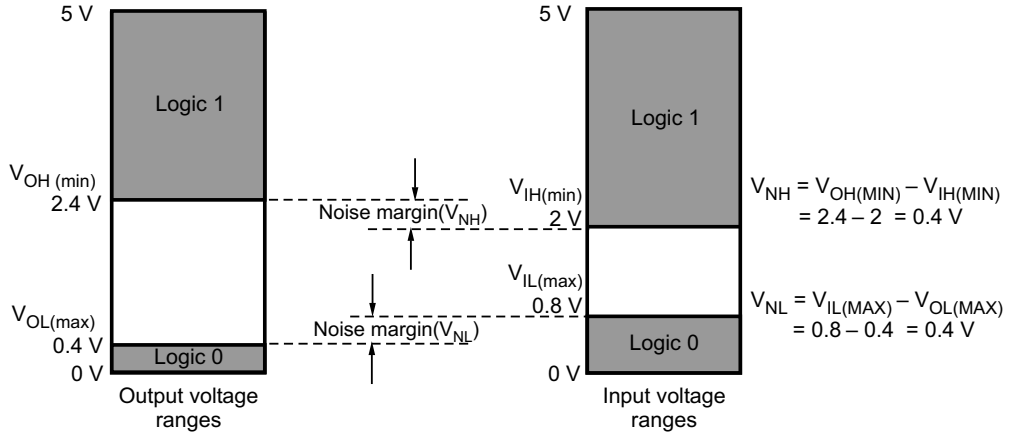


Fig. 3.3.13 TTL logic levels and noise margin

Power dissipation and propagation delay

A standard TTL gate has an average power dissipation of about 10 mW. It may vary from this value because of signal levels, tolerances etc. We know that, the propagation delay time is the time it takes for the output of a gate to change after the inputs have changed. The propagation delay time of a TTL gate is approximately 10 nanoseconds.

Fan-out

A standard TTL output can typically drive 10 standard TTL inputs. Therefore, standard TTL has fan-out 10.

Table 3.3.6 summarizes the characteristics of standard TTL

Characteristics	Values
Supply voltage	For 74 series - (4.75 to 5.25) units For 54 series - (4.5 to 5.5) units
Temperature range	For 74 series - (0 °C to 70 °C) For 54 series - (– 55 °C to 125 °C)
Voltage levels	$V_{OL\,(max)}$ - 0.4 V $V_{OH\,(min)}$ - 2.4 V $V_{IL(max)}$ - 0.8 V $V_{IH\ (min)}$ - 2.0 V
Noise margin	0.4 V
Power dissipation	10 mW per gate
Propagation delay	Typically 10 ns
Fan-out	10

Table 3.3.6 Standard TTL characteristics

3.3.9 Advantages and Disadvantages of TTL Family

Advantages of TTL

1. High speed operation. Fastest among the saturated logic families. The propagation delay time is about 10 ns.
2. Moderate power dissipation.
3. Available in commercial and military versions.
4. Available for wide range of functions.
5. Low cost.
6. Moderate packaging density.

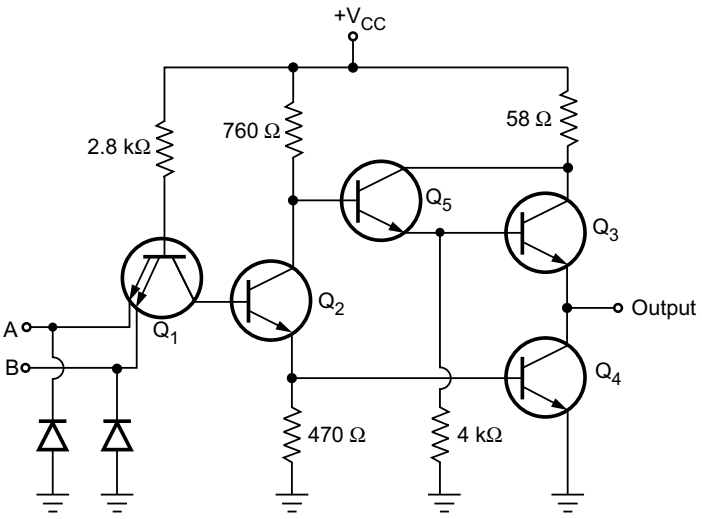
Disadvantages of TTL

1. Higher power dissipation than CMOS.
2. Lower noise immunity than CMOS.
3. Less fan-out than CMOS.

3.3.10 High-Speed (H) and Low Power (L) TTL

The 74L and 74H series were developed to provide low-power and high-speed version of TTL, respectively. Both have the same basic circuit as the standard 74 series, but differences in circuit component values and one or more additional component give these two series their different characteristics.

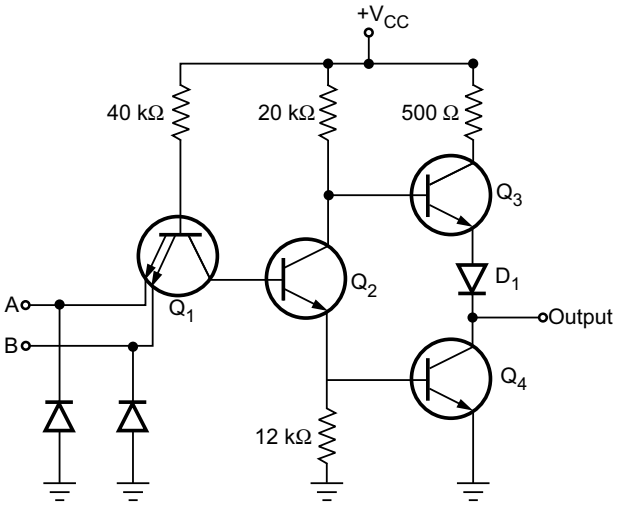
Fig. 3.3.14 (a) shows 2-input high speed TTL NAND gate. Besides smaller resistance values, the only difference between this gate and the standard version is the additional transistor Q_5. This transistor acts as a driver for Q_3. Actually Q_3 and Q_5 form a darlington pair, which has much more current gain. This results in higher switching speed of the output from low to high. The switching time is also reduced due to low resistance values. However,



(a) 2-input high speed NAND gate

due to reduced resistance values power consumption is more. It is important to note that the base-emitter junctions of Q_3 and Q_5 raise the threshold level at the base of Q_5 to 1.4 V and therefore, the darlington connection eliminates the need for a diode in the totem-pole.

Fig. 3.3.14 (b) shows 2-input low power TTL NAND gate. The only difference between this gate and the standard version is the much larger resistance values, which reduce power consumption at the expense of switching speed. The low-power series consumes less power of the order of 1 mW with longer propagation delay around 33 ns. On the other hand high-speed series has a reduced propagation delay of about 6 ns at the cost of higher power consumption around 32 mW.



(b) 2-input low power NAND gate

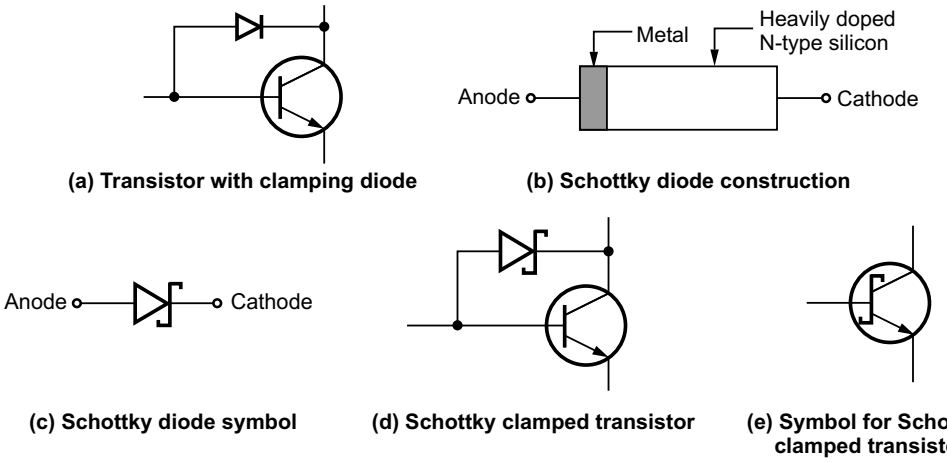
Fig. 3.3.14 74H00 and 74L00 NAND gates

3.3.11 Schottky TTL

With standard TTL, high-speed TTL and low-power TTL, when transistors are ON, they are driven into hard saturation. This causes a surplus of carriers to be stored in the base. This excess charge carriers in the base region must be removed before the transistors is switched from ON to OFF. The time required to remove these carriers, called storage time, is responsible for high propagation delays or low switching times. One way to counter this problem is to prevent the transistor from going into deep saturation when it is turned ON i.e. to restrict the forward bias of its base-to-collector junction to few tenths of a volt. This can be accomplished by connecting a diode across the base-to-collector junction, as shown in the Fig. 3.3.15 (a). The diode does not allow to increase the forward bias voltage at base-to-collector junction above its cut-in voltage. In other words, it clamps base-to-collector voltage upto its small cut-in voltage. As a result the junction cannot be heavily forward biased and the transistor is kept out of deep saturation.

We know that germanium diodes have less cut-in voltage but they cannot be fabricated in silicon integrated circuits. The Schottky diode, consisting of a silicon-metal junction with a cut-in voltage of 0.3 to 0.4 volts is ideal for this purpose.

Fig. 3.3.15 (b) and (c) show the construction, and Fig. 3.3.15 (d) and (e) show Schottky clamped NPN transistor and its symbol.



(a) Transistor with clamping diode (b) Schottky diode construction

(c) Schottky diode symbol (d) Schottky clamped transistor (e) Symbol for Schottky clamped transistor

Fig. 3.3.15

Another important parameter of Schottky diode is that it has very little capacitance and fast recovery time. So it can be switched rapidly without storage time delays.

3.3.11.1 Schottky (S) and Low-Power Schottky (LS) TTL

Fig. 3.3.16 shows 2-input Schottky NAND gate. Looking at the Fig. 3.3.16, we can notice that it uses smaller resistor values to help improve switching times. This increases the device average power dissipation to about 20 mW, about the same as for 74H series.

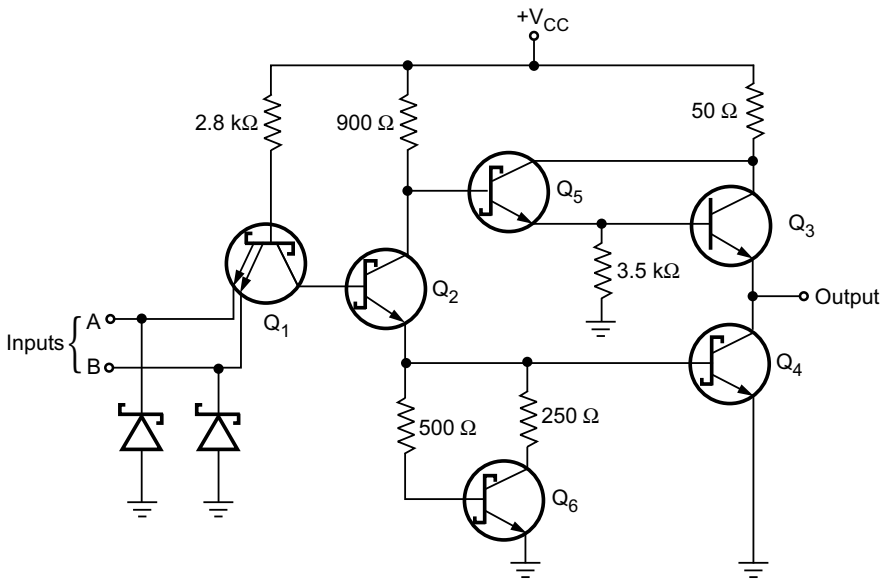


Fig. 3.3.16 2-input Schottky NAND gate

Note that all the transistors used are Schottky clamped transistors, except Q_3, which does not saturate. This series also use a darlington pair (Q_3 and Q_5) to provide shorter output rise time when switching from OFF to ON. One more transistor (Q_6) is used in this series. This regulates the current flow into the base of Q_4 and helps in turning Q_4 off rapidly.

The Fig. 3.3.17 shows the circuit diagram for a two-input LS-TTL NAND gate (74LS00). We know that the NAND function can be obtained by inverting the output of AND function. The circuit uses same logic to implement NAND function. The circuit is basically divided into three functional parts.

- Diode AND gate and input protection
- Phase shifter.
- Output stage.

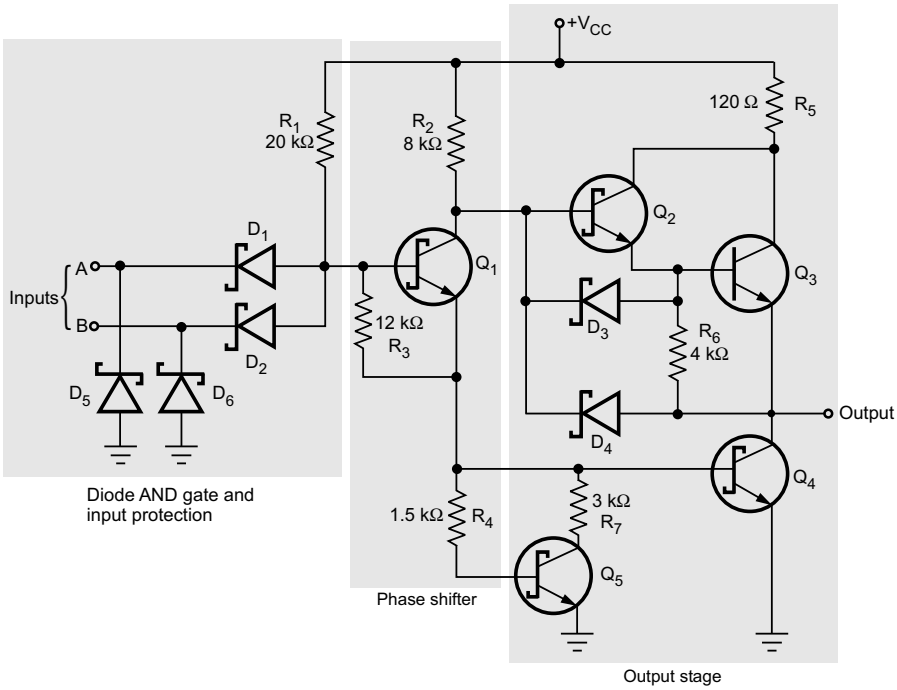


Fig. 3.3.17 2-input low power Schottky NAND gate

As shown in the Fig. 3.3.18 the diodes D_1 and D_2 with resistor R_1 forms the two input AND gate. The diodes D_5 and D_6 at the input are used as protective diodes. These diodes protect the circuit from large negative transients on input lines. If an input attempts to go more than about 1 V negative, the protective diode conducts and clamps the input voltage to its conducting voltage.

The transistor Q_2 acts as an inverter. The Q_2 alongwith the surrounding resistors form a **phase splitter** that controls the output stage. The diode AND gate and the phase splitter, thus represents the NAND function.

The output stage has two basic transistors Q_4 and Q_5, only one of which is ON at any time. The TTL output stage is called a **totem-pole** or **push-pull output**. Similar to the p-channel and n-channel transistors in CMOS, Q_4 and Q_5 provide active pull-up and pull-down to HIGH and LOW states, respectively.

It is important to note that all the transistors used are Schottky clamped transistors, except Q_4, which does not saturate. The circuit uses a darlington pair (Q_3 and Q_4) to provide shorter output rise time when switching from OFF to ON. The transistor Q_6 connected in series regulates the current flow into the base of Q_5 and helps in turning Q_5 OFF rapidly.

When TTL output is changing from HIGH to LOW or vice-versa, there is a short time when both transistors may be ON. During this time resistor R_5 limits the amount of current flowing from V_{CC} to ground.

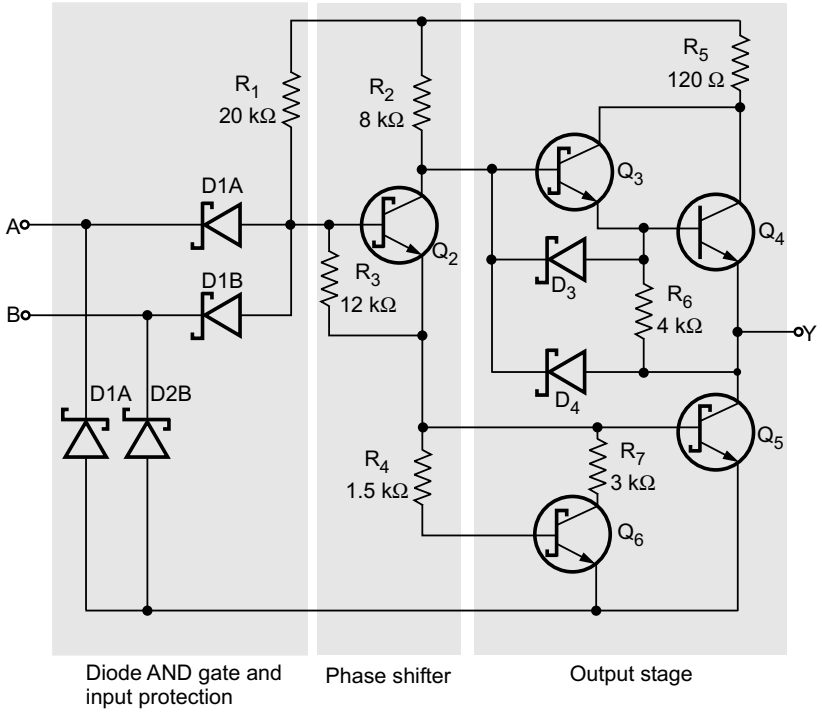


Fig. 3.3.18 Circuit diagram of two-input LS-TTL NAND gate

The Fig. 3.3.19 shows the functional operation of a TTL two-input NAND gate. It also shows the truth table and logic symbol.

A	B	Q_2	Q_3	Q_4	Q_5	Q_6	Y
0	0	OFF	ON	ON	OFF	OFF	1
0	1	OFF	ON	ON	OFF	OFF	1
1	0	OFF	ON	ON	OFF	OFF	1
1	1	ON	OFF	OFF	ON	ON	0

(a) Function table

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

(b) Truth table

(c) Logic symbol

Fig. 3.3.19

3.3.12 Comparison of TTL Series Characteristics

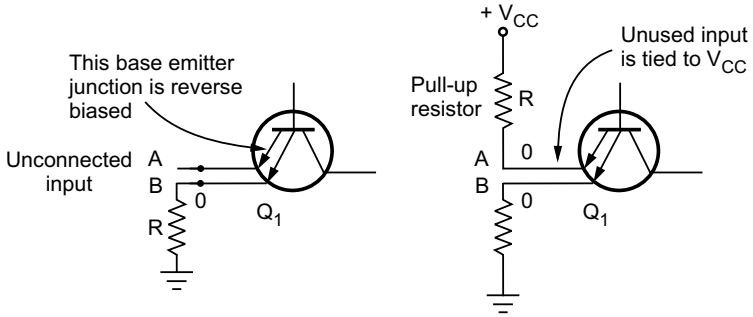
Table 3.3.7 shows the typical values of characteristics of all the TTL series. All of the performance ratings, except for maximum clock rate, are for a NAND gate in each series.

Parameter	74	74S	74LS	74AS	74ALS	74F
Propagation delay (ns)	9	3	9.5	1.7	4	3
Power dissipation (mW)	10	20	2	8	1.2	6
Speed-power product (pJ)	90	60	19	13.6	4.8	18
Max. clock rate (MHz)	35	125	45	200	70	100
Fan-out	10	10 (Low level leads) 20 (High level leads)	20	40	20	33
Voltage parameters :						
$V_{OH\,(min)}$(V)	2.4	2.7	2.7	2.5	2.5	2.5
$V_{OL\,(max)}$(V)	0.4	0.4	0.5	0.5	0.4	0.5
$V_{IH\,(min)}$(V)	2.0	2.0	2.0	2.0	2.0	2.0
$V_{IL\,(max)}$(V)	0.8	0.8	0.8	0.8	0.8	0.8

Table 3.3.7

3.3.13 Unconnected Inputs

If any input of TTL gate is left open or unconnected, then the corresponding base emitter junction of the input transistor Q_1 is reversed bias. See Fig. 3.3.20 (a). Therefore, the open or unconnected input behaves exactly same way as if a logic 1 is applied to that input. Thus, in TTL ICs all the unconnected inputs are treated as logic 1. However, it is recommended that the unused inputs should be either connected to some used input or tied to V_{CC} through pull-up resistor, as shown in the Fig. 3.3.20 (b).



(a) Unconnected input **(b) Standard connection**

Fig. 3.3.20

Review Questions

1. *Explain the concept, operation and characteristics of TTL family.* **Dec.-10, Marks 8**
2. *Write a short note on TTL family.* **Dec.-07, Marks 8**
3. *Draw the TTL inverter circuit.* **Dec.-04, May-12, Marks 10**
4. *Explain the working of 2 input TTL totem-pole NAND gate circuit.* **Dec.-04, 05, May-08, 09, Marks 8**
5. *Explain the operation of 3-input TTL NAND Gate with required diagram and truth table.* **May-08, Marks 8**
6. *Explain the working of a basic totem-pole TTL two input NAND gate.* **May-03, 05, Dec.-03, Marks 10; May-09, Marks 2**
7. *Draw the circuit diagram of TTL open collector two input NAND gate.* **May-09, Marks 2**
8. *Write a purpose of Pull-up resistor in TTL logic family.*
9. *What is the difference between totem-pole output and open-collector output in TTL logic family.*
10. *Draw the circuit diagram and explain the working of TTL inverter with tristate output.* **Dec.-04, 05, May-06, 09, Marks 8**
11. *What is meant by tri-state capability ?* **May-07, Marks 2**
12. *Define tri-states.* **Dec.-03, 10, May-05, Marks 2**
13. *Draw the circuit diagram of a two input TTL NAND gate with tristate output and explain its action, clearly showing logic and voltage levels.* **Dec.-12, Marks 16**
14. *Discuss about the TTL parameters.* **Dec.-04, Marks 10**

15. *State the important characteristics of TTL family.* **May-10, Dec.-10, Marks 2**
16. *Name and explain the characteristics of TTL family.* **May-06, Marks 8**
17. *Distinguish between 7400 series and 5400 series.* **May-04, Marks 2**
18. *State advantages and disadvantages of TTL family.*
19. *List the advantages of TTL logic family.*
20. *What is the advantage of Schottky TTL family?* **May-04, Marks 2**
21. *Explain 2-input high speed TTL NAND gate.*
22. *Explain 2-input Low speed TTL NAND gate.*
23. *Explain 2-input Schottky and Low-power Schottky TTL NAND gate.*

3.4 CMOS Logic

May-03, 04, 06, 07, 08, 10, 11, Dec.-03, 04, 06, 07, 10

Digital circuits with MOSFETs can be grouped into three categories :

- PMOS - uses only P-channel enhancement MOSFETs,
- NMOS - uses only N-channel enhancement MOSFETs, and
- CMOS (complementary MOS) - uses both P and N-channel devices.

PMOS and NMOS digital ICs are economical than CMOS ICs because they have greater packing density than CMOS. NMOS has twice the packaging density than PMOS. Further more, NMOS can operate at about three times faster than their PMOS counterparts. This is because NMOS has faster moving current carriers (electrons) whereas PMOS has slower moving current carriers (holes). CMOS has the greatest complexity and lowest packaging density; however, it has important advantages of high speed and much lower power dissipation. NMOS and CMOS are widely used in the digital ICs, but PMOS ICs are no longer part of new designs.

CMOS circuits contain both NMOS and PMOS devices to speed the switching of capacitive loads. It consumes low power and can be operated at high voltages, resulting in improved noise immunity.

3.4.1 CMOS Inverter

Fig. 3.4.1 shows the basic CMOS inverter circuit. It consists of two MOSFETs in series in such a way that the p-channel device has its source connected to $+ V_{DD}$ (a positive voltage) and the n-channel device has its source connected to ground. The gates of the two devices are connected together as the common input and the drains are connected together as the common output.

1. When input is HIGH, the gate of Q_1 (p-channel) is at 0 V relative to the source of Q_1 i.e. $V_{gs_1} = 0\,V$. Thus, Q_1 is OFF. On the other hand, the gate of Q_2 (n-channel) is at $+V_{DD}$ relative to its source i.e. $V_{gs_2} = +V_{DD}$. Thus, Q_2 is ON. This will produce $V_{OUT} \approx 0$ V, as shown in the Fig. 3.4.2 (a).

2. When input is LOW, the gate of Q_1 (p-channel) is at a negative potential relative to its source while Q_2 has V_{gs} = 0 V. Thus, Q_1 is ON and Q_2 is OFF. This produces output voltage approximately + V_{DD}, as shown in the Fig. 3.4.2 (b).

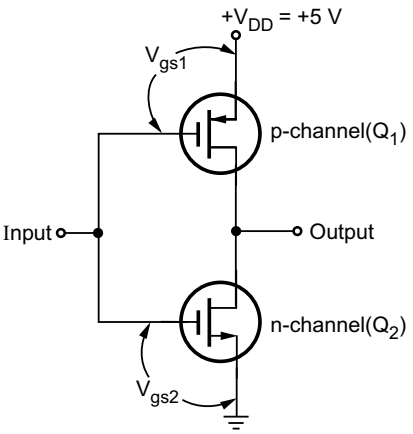
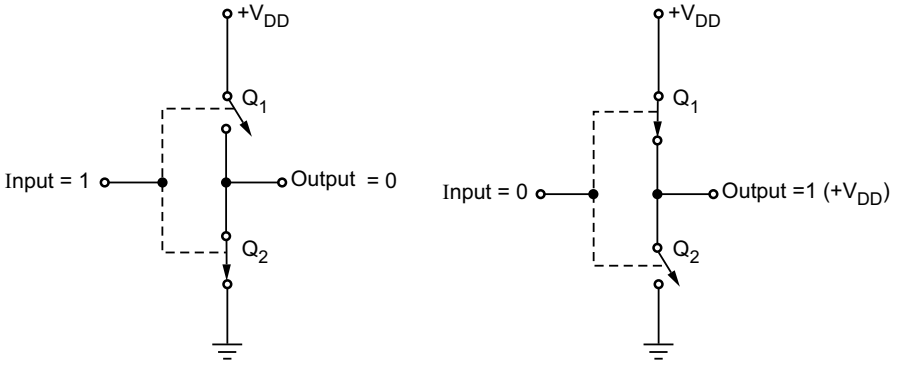


Fig. 3.4.1 CMOS inverter circuit



(a) Input = 1 **(b) Input = 0**

Fig. 3.4.2 Operation of CMOS inverter for both input conditions

Table 3.4.1 summarizes the operation of CMOS inverter circuit.

The Fig. 3.4.3 shows, different symbols used for the p-channel and n-channel transistors to reflect their logical behaviour. The n-channel transistor (Q_2) is switched 'ON' when a HIGH voltage is applied at the input. The p-channel transistor (Q_1) has the opposite behaviour, it is switched ON when a LOW voltage is applied at the input. It is indicated by placing bubble in the symbol.

A	Q_1	Q_2	Output
0	ON	OFF	1
1	OFF	ON	0

Table 3.4.1 Truth table of inverter

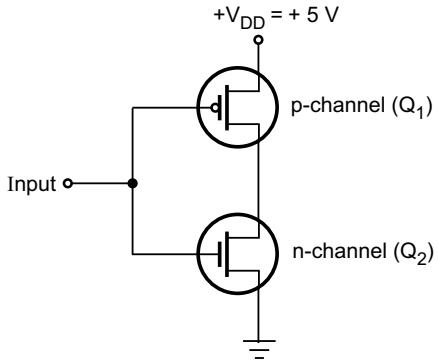


Fig. 3.4.3 The CMOS inverter

The Fig. 3.4.4 shows the transfer characteristics of CMOS inverter.

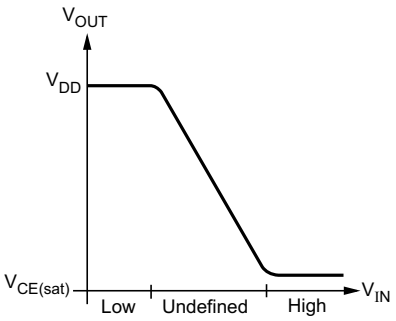


Fig. 3.4.4 CMOS inverter transfer characteristics

3.4.2 CMOS NAND Gate

Fig. 3.4.5 shows CMOS 2-input NAND gate. It consists of two p-channel MOSFETs, Q_1 and Q_2, connected in parallel and two n-channel MOSFETs, Q_3 and Q_4 connected in series.

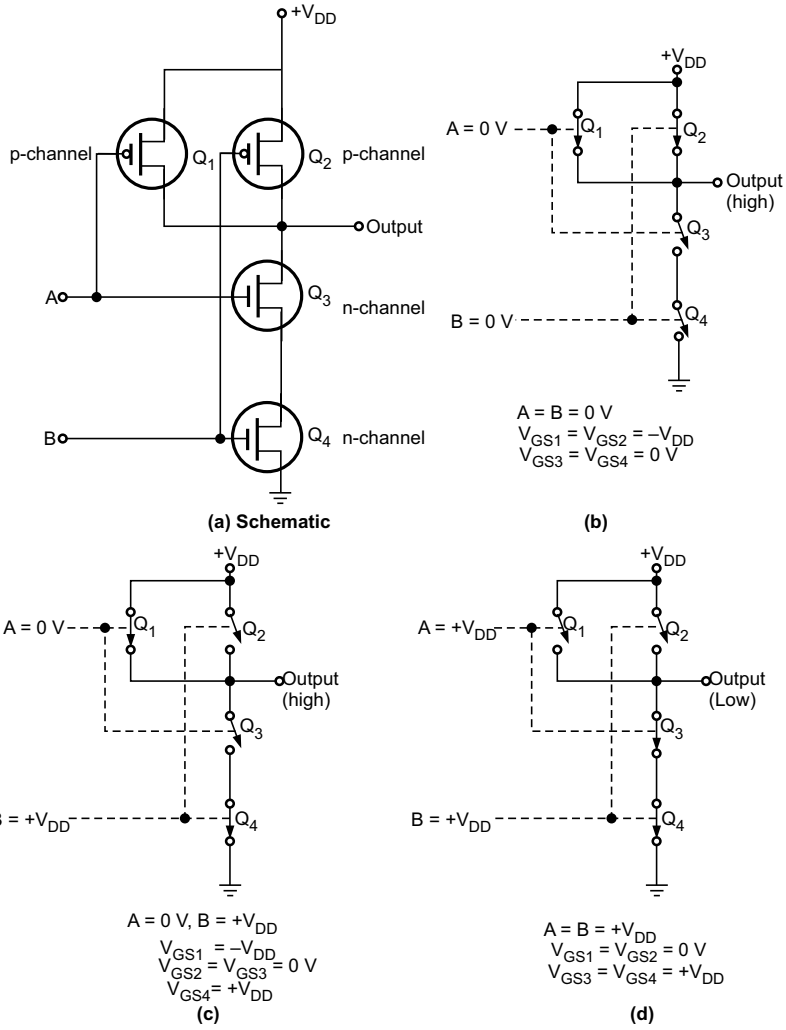


Fig. 3.4.5 CMOS NAND gate

Fig. 3.4.5 (a) shows the equivalent switching circuit when both inputs are low. Here, the gates of both p-channel MOSFETs are negative with respect to their sources, since the sources are connected to $+V_{DD}$. Thus, Q_1 and Q_2 are both ON. Since the gate-to-source voltages of Q_3 and Q_4 (n-channel MOSFETs) are both 0 V, those MOSFETs are OFF. The output is therefore connected to $+V_{DD}$ (HIGH) through Q_1 and Q_2 and is disconnected from ground, as shown in the Fig. 3.4.5 (b). Fig. 3.4.5 (c) shows the equivalent switching circuit when A = 0 and B = $+V_{DD}$. In this case, Q_1 is on because $V_{GS_1} = -V_{DD}$ and Q_4 is ON because $V_{GS_4} = +V_{DD}$. MOSFETs Q_2 and Q_3 are off because their gate-to-source voltages are 0 V. Since Q_1 is ON and Q_3 is OFF, the output is connected to $+V_{DD}$ and it is disconnected from ground. When A = $+V_{DD}$ and B = 0 V, the situation is similar (not shown); the output is connected to $+V_{DD}$ through Q_2 and it is disconnected from ground because Q_4 is OFF. Finally, when both inputs are high (A = B = $+V_{DD}$), MOSFETs Q_1 and Q_2 are both OFF and Q_3 and Q_4 are both ON. Thus, the output is connected to the ground through Q_3 and Q_4 and it is disconnected from + V_{DD}. The Table 3.4.2 summarizes the operation of 2-input CMOS NAND gate.

A	B	Q_1	Q_2	Q_3	Q_4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	1
1	0	OFF	ON	ON	OFF	1
1	1	OFF	OFF	ON	ON	0

Table 3.4.2 Truth table of NAND gate

Note : p-channel MOSFET is ON when its gate voltage is negative with respect to its source whereas n-channel MOSFET is ON when its gate voltage is positive with respect to its source

The Fig. 3.4.6 shows the circuit diagram, function table and logic symbol of CMOS 3-input NAND gate.

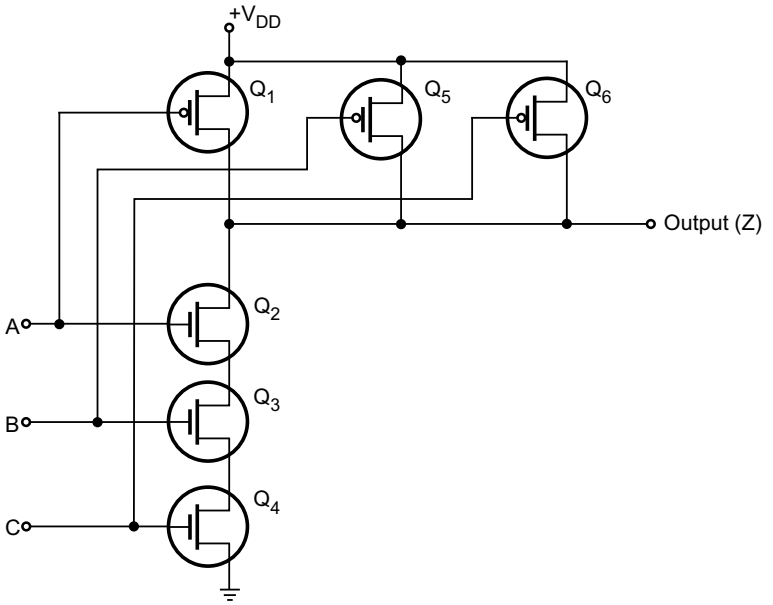


Fig. 3.4.6 (a) Circuit diagram

A	B	C	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	Z
0	0	0	ON	OFF	OFF	OFF	ON	ON	1
0	0	1	ON	OFF	OFF	ON	ON	OFF	1
0	1	0	ON	OFF	ON	OFF	OFF	ON	1
0	1	1	ON	OFF	ON	ON	OFF	OFF	1
1	0	0	OFF	ON	OFF	OFF	ON	ON	1
1	0	1	OFF	ON	OFF	ON	ON	OFF	1
1	1	0	OFF	ON	ON	OFF	OFF	ON	1
1	1	1	OFF	ON	ON	ON	OFF	OFF	0

Fig. 3.4.6 (b) Function table

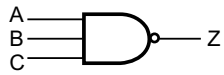


Fig. 3.4.6 (c) Logic symbol

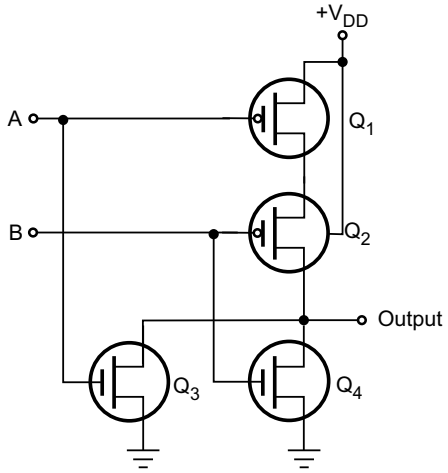
3.4.3 CMOS NOR Gate

Fig. 3.4.7 shows 2-input CMOS NOR gate. Here, p-channel MOSFETs Q_1 and Q_2 are connected in series and n-channel MOSFETs Q_3 and Q_4 are connected in parallel.

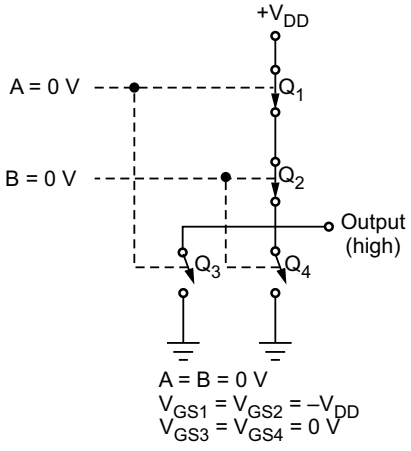
A	B	Q_1	Q_2	Q_3	Q_4	Output
0	0	ON	ON	OFF	OFF	1
0	1	ON	OFF	OFF	ON	0
1	0	OFF	ON	ON	OFF	0
1	1	OFF	OFF	ON	ON	0

Table 3.4.3 Truth table for NOR gate

Like NAND circuit, this circuit can be analyzed by realizing that a LOW at any input turns ON its corresponding p-channel MOSFET and turns OFF its corresponding n-channel MOSFET, and vice versa for a HIGH input. This is illustrated in Fig. 3.4.7. The Table 3.4.3 summarizes the operation of 2-input NOR gate.



(a) Schematic



(b)

Fig. 3.4.7

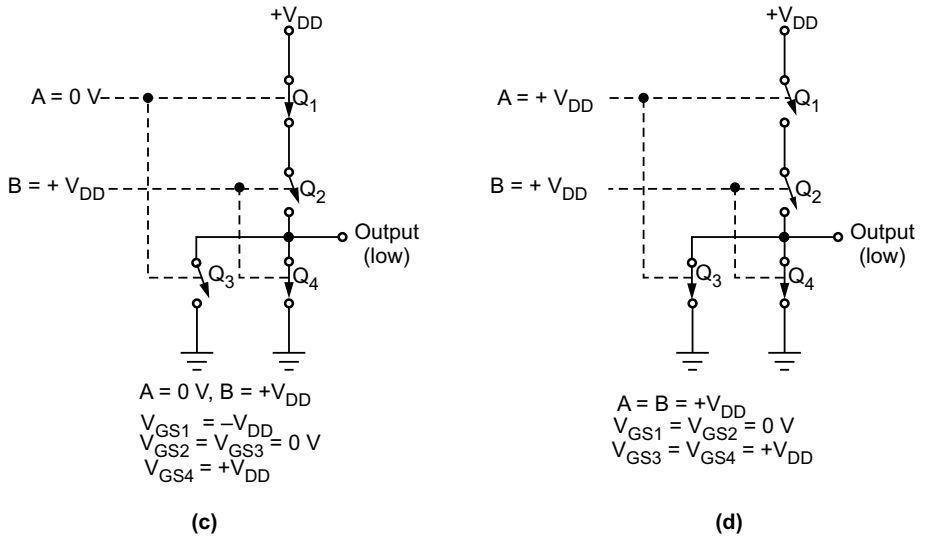


Fig. 3.4.7 CMOS NOR gate

3.4.4 Characteristics of CMOS Family

Operating Speed : Slower than TTL series. Approximately 25 to 100 ns depending on the subfamily of CMOS. It also depends on the power supply voltage.

Voltage Levels and Noise Margins : The voltage levels for CMOS varies according to their subfamilies. These are listed in Table 3.4.4. Noise margins in table are calculated as follows.

Parameter	CMOS series				
	4000 B	74 HC	74 HCT	74 AC	74 ACT
$V_{IH(min)}$	3.5	3.5	2.0	3.5	2.0
$V_{IL(max)}$	1.5	1.0	0.8	1.5	0.8
$V_{OH(min)}$	4.95	4.9	4.9	4.9	4.9
$V_{OL(max)}$	0.05	0.1	0.1	0.1	0.1
V_{NH}	1.45	1.4	2.9	1.4	2.9
V_{NL}	1.45	0.9	0.7	1.4	0.7

Table 3.4.4

$$V_{NH} = V_{OH(min)} - V_{IH(min)}$$

$$V_{NL} = V_{IL(max)} - V_{OL(max)}$$

Fan-out : The CMOS inputs have an extremely large resistance (10^{12} Ω) that draws essentially no current from the signal source. Each CMOS input, however, typically presents a 5 pF load to ground as shown in the Fig. 3.4.8. This input capacitance limits the number of CMOS inputs that one CMOS output can drive.

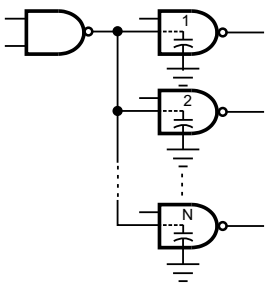


Fig. 3.4.8 One CMOS output driving several CMOS inputs

The CMOS output has to charge and discharge the parallel combination of all the input capacitances. This charging and discharging time increases as we increase number of loads. Typically, each CMOS load increases the driving circuit's propagation delay by 3 ns. Thus, fan-out for CMOS depends on the permissible maximum propagation delay. Typically, CMOS outputs are limited to a fan-out of 50 for low-frequency operation (≤ 1 MHz). Of course, for high-frequency operation the fan-out would have to be less.

Power Dissipation (P_D) : The power dissipation of a CMOS IC is very low as long as it is in a d.c. condition. Unfortunately, power dissipation of CMOS IC increases in proportion to the frequency at which the circuits are switching states. For example, a CMOS NAND gate that has P_D =10 nW under d.c. conditions will have P_D = 0.1 mW at a frequency of 100 kHz and 1 mW at 1 MHz.

When CMOS output switches from LOW to HIGH, a transient charging current has to be supplied to the load capacitance. Therefore, as the switching frequency increases, the average current drawn from V_{DD} also increases, resulting increase in power dissipation.

Propagation Delay : The propagation delay in CMOS is the sum of delay due to internal capacitance and due to load capacitance. The delay due to internal capacitance is called the intrinsic propagation delay. The delay due to load capacitance can be approximated as follows.

$$t_p(C_L) \approx 0.5\ R_o\ C_L \text{ seconds}$$

where $t_p(C_L)$ is either t_{pLH} or $t_{pHL.}$

R_o is the output resistance of the gate and C_L is the total load capacitance.

The R_o depends on the supply voltage and it can be approximated as

$$R_o \approx \frac{V_{CC}}{I_{os}}$$

where I_{os} is the short circuit output current.

Unused Inputs : CMOS inputs should never be left disconnected. All CMOS inputs have to be tied either to a fixed voltage level (0 V or V_{DD}) or to another input. This rule applies even to the inputs of extra unused logic gates on a chip. An unused CMOS input is susceptible to noise and static charges that could easily bias both the P and N-channel MOSFETs in the conductive state, resulting in increased power dissipation and possible overheating.

Static-charge Susceptibility (CMOS Hazards) : Every CMOS device is vulnerable to the building up of electrical charge on its insulated gate. Recall that the relationship between charge Q and voltage V on a capacitor having capacitance C is

$$V = \frac{Q}{C}$$

Since the input capacitance at the gate is usually quite small (a few picofarads), a relatively small amount of charge can create a large voltage which may be greater than the breakdown voltage of a MOS gate (typically 100 V).

The primary source of charge is "static" electricity, usually produced by handling and the motion of various kinds of plastics and textiles. The CMOS devices are protected against this static charge by on-chip diode-resistor network, as shown in the Fig. 3.4.9. These diodes are designed to turn ON and limit the size of the input voltage to well below any damaging value.

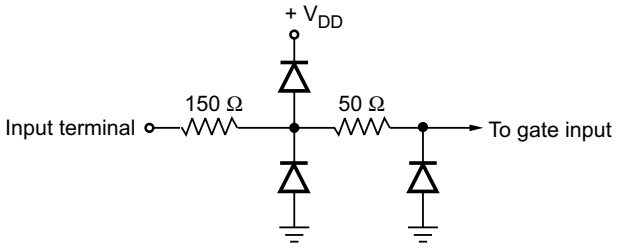


Fig. 3.4.9 Typical network used to protect CMOS from static charges

Latch-up : CMOS integrated circuits contain parasitic PNP and NPN transistors : transistors that exist because of the proximity of P and N materials embedded in the substrate. Their existence is not intentional but is unavoidable. Because of conducting paths between a pair of such transistors, a device can be triggered into a heavy conduction mode, known as **latch-up**. This heavy conduction mode, results large current flow which can destroy IC. Most CMOS circuits contain protective measures to prevent latch-up, but it can still occur if the manufacturers specified maximum ratings are exceeded.

3.4.5 Advantages and Disadvantages of CMOS Family

Advantages

1. Consumes less power.
2. Can be operated at high voltages, resulting in improved noise immunity.
3. Fan-out is more.
4. Better noise margin.

Disadvantages

1. Susceptible to static charge.
2. Switching speed low.
3. Greater propagation delay.

Review Questions

1. *Explain the concept, operation and characteristics of CMOS technology.* **Dec.-10, Marks 8**
2. *Write a short note on CMOS family.* **Dec.-07, Marks 8**
3. *Draw the schematic and explain the operation of a CMOS inverter. Also explain its characteristics.* **May-11, Marks 8**
4. *Draw the circuit of a CMOS two input NAND gate and explain its operation.* **May-03, Dec.-03, 04, May-06, 08, Marks 8**
5. *Draw and explain CMOS NAND gate.*
6. *Draw and explain the circuit diagram of a CMOS NOR gate.* **Dec.-03, 04, May-04, 08, Marks 12**
7. *Explain the characteristics of CMOS family.* **May-06, 07, 10, Dec.-06, Marks 8**
8. *State the advantages and disadvantages of CMOS family.* **Dec.-03, Marks 2**

3.5 Comparison between TTL and CMOS Families **Dec.-06, May-07, 09**

Sr. No.	Parameter	CMOS	TTL
1.	Device used	n-channel and p-channel MOSFET	Bipolar junction transistor
2.	$V_{IH(min)}$	3.5 V	2 V
3.	$V_{IL(max)}$	1.5 V	0.8 V
4.	$V_{OH(min)}$	4.95 V	2.7 V
5.	$V_{OL(max)}$	0.005 V	0.4 V
6.	High level noise margin	V_{NH} = 1.45 V	0.4 V
7.	Low level noise margin	V_{NL} = 1.45 V	0.4 V
8.	Noise immunity	Better than TTL	Less than CMOS
9.	Propagation delay	70 ns	10 ns
10.	Switching speed	Less than TTL	Faster than CMOS

11.	Power dissipation per gate	0.1 mW	10 mW
12.	Speed power product	0.7 pJ	100 pJ
13.	Fan-out	50	10
14.	Power supply voltage	3 - 15 V	Fixed 5 V
15.	Power dissipation	Increase with frequency	Increase with frequency
16.	Application	Portable instrument where battery supply is used.	Laboratory instruments.

Table 3.5.1 Comparison between TTL and CMOS families

Review Question

1. *What is the major difference between TTL and CMOS.*

Dec.-06, May-07, 09, Marks 8

3.6 Interfacing TTL and CMOS Families

Interfacing means connecting the output(s) of one circuit or system to the input(s) of another circuit or system that may have different electrical characteristics. When two circuits have different electrical characteristics direct connection cannot be made. In such cases driver and load circuits are connected through interface circuit. Its function is to take the driver output signal and condition it so that it is compatible with requirements of the load.

One must consider following important points while interfacing two circuits or systems.

The driver output must satisfy the voltage and current requirements of the load circuit.

The driver and load circuit may require different power supplies. In such cases the output of both circuit must swing between its specified voltage ranges.

3.6.1 TTL Driving CMOS

Here, TTL is a driver circuit and CMOS is a load circuit. The two circuits are from different families with different electrical characteristics. Therefore, we must check that the driving device can meet the current and voltage requirements of the load device.

	CMOS		TTL		
	4000B	**74HC/HCT**	**74**	**74LS**	**74AS**
$I_{IH\ (max)}$	1 μA	1 μA	40 μA	20 μA	200 μA
$I_{IL\ (max)}$	1 μA	1 μA	1.6 mA	0.4 mA	2 mA
$I_{OH\ (max)}$	0.4 mA	4 mA	0.4 mA	0.4 mA	2 mA
$I_{OL\ (max)}$	0.4 mA	4 mA	16 mA	8 mA	20 mA

Table 3.6.1 Input/output currents for standard devices with supply voltage of 5 V

Table 3.6.1 indicates that the input current values for CMOS are extremely low compared with the output current capabilities of any TTL series. Thus, TTL has no problem meeting the CMOS input current requirements.

But when we compare the TTL output voltages with the CMOS input voltage requirements we find that :

$V_{OH\ (min)}$ for TTL << $V_{IH\ (min)}$ for CMOS for these situations TTL output must be raised to an acceptable level for CMOS. This can be done by connecting pull-up resistor at the output of TTL, as shown in the Fig. 3.6.1. The pull-up resistor causes the TTL output to rise to approximately 5 V in the HIGH state, thereby providing an adequate CMOS input voltage level.

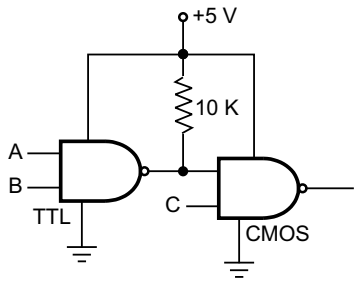


Fig. 3.6.1 TTL driving CMOS using external pull-up resistor

TTL Driving HIGH Voltage CMOS : When output CMOS circuit is operating with V_{DD} greater than 5 V, the situation becomes more difficult. The outputs of many TTL devices cannot be operated at more than 5 V. In such cases some alternative arrangements are made. Two of them are discussed below :

1. When the TTL output cannot be pulled up to V_{DD}, one can use open collector buffer as an interface between totem-pole TTL output and CMOS operating at V_{DD} > 5 V, as shown in the Fig. 3.6.2.

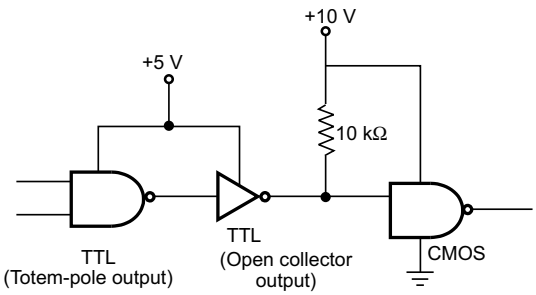


Fig. 3.6.2 Open collector buffer used as interface circuit

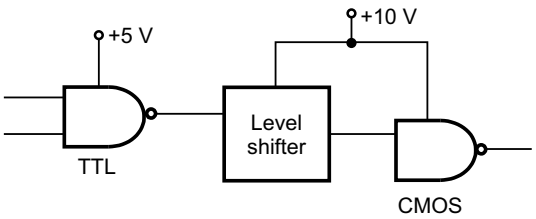


Fig. 3.6.3 Level shifter used as interface circuit

2. The second alternative is to use **level translator** circuit, such as the 40104. This is a CMOS chip that is designed to take a low-voltage input (e.g. from TTL) and translate it to high voltage output for CMOS. Fig. 3.6.3 shows the circuit arrangement.

3.6.2 CMOS Driving TTL

Before we consider the problem of interfacing CMOS outputs to TTL inputs, it will be helpful to review the CMOS output and TTL input characteristics for the two logic states.

For CMOS (4000B)	For TTL
$V_{OH(min)}$: 4.95 V	$V_{IH(min)}$: 2.0 V
$V_{OL(max)}$: 0.05 V	$V_{IL(max)}$: 0.8 V
$I_{OH(max)}$: 0.4 mA	$I_{IH(max)}$: 40 μA
$I_{OL(max)}$: 0.4 mA	$I_{IL(max)}$: 1.6 mA

Table 3.6.2

CMOS Driving TTL in the HIGH state : Above voltage parameters show that CMOS outputs can easily supply enough voltage (V_{OH}) to satisfy the TTL input requirement in the HIGH state (V_{IH}). The parameters also show that CMOS outputs can supply more than enough current (I_{OH}) to meet the TTL input current requirements (I_{IH}). Thus no special consideration is required for CMOS driving TTL in the HIGH state.

CMOS Driving TTL in LOW state : The parameters in the Table 3.6.2 show that CMOS output voltage (V_{OL}) satisfies TTL input requirement in the LOW state (V_{IL}). However, the current requirements in the LOW state are not satisfied. The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input of the TTL. In such situations some type of interface circuit is needed between the CMOS and TTL devices. Fig. 3.6.4 shows the possible interface circuit.

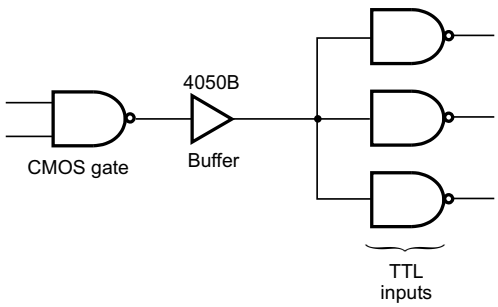


Fig. 3.6.4 CMOS driving TTL in LOW state using buffer

In Fig. 3.6.4 the CMOS 4050B, non-inverting buffer is used as an interfacing circuit. It has an output current rating of $I_{OL\ (max)}$ = 3 mA which satisfies the TTL input current requirement.

HIGH Voltage CMOS Driving TTL : Some IC manufacturers have provided several 74LS TTL devices that can withstand input voltages as high as 15 V. These devices can be driven directly from CMOS outputs operating at V_{DD} = 15 V. However, most TTL inputs cannot handle more than 7 V and so interface is necessary if they are to be driven from high-voltage CMOS. In such situations **voltage level translators** are used. They convert the high-voltage input to a 5 V output that can be connected to TTL. Fig. 3.6.5 shows how the 4050B can be used to perform this level translation between 15 V and 5 V.

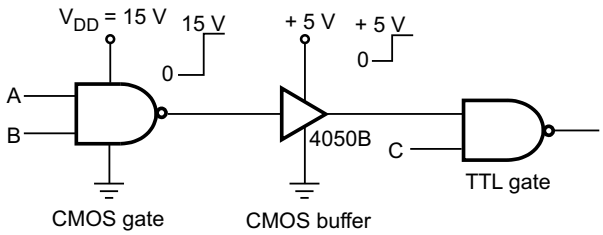


Fig. 3.6.5 Level translation using CMOS buffer

Example 3.6.1 *Why is a buffer required between some CMOS outputs and TTL inputs ?*

Solution : The current requirement of CMOS output at LOW state are not statisfied by TTL input. The TTL input has a relatively high input current in the LOW state (1.6 mA) and CMOS output current at LOW state (I_{OL}) is not sufficient to drive even one input of the TTL. Hence buffer is used for interfacing.

Example 3.6.2 *Which CMOS series can have its inputs driven directly from a TTL output ? Justify your answer.*

Solution : VHC and VHCT CMOS families are TTL compatible.

Review Question

1. *Explain with neat diagram interfacing of a TTL gate driving CMOS gates and vice-versa.*

3.7 Two Marks Questions with Answers

Q.1 ***Classify the basic families that belong to the bipolar families and to the MOS families.*** **May-06**

Ans. : The Fig. 3.7.1 gives the classification of families that belong to the bipolar families and to the MOS families.

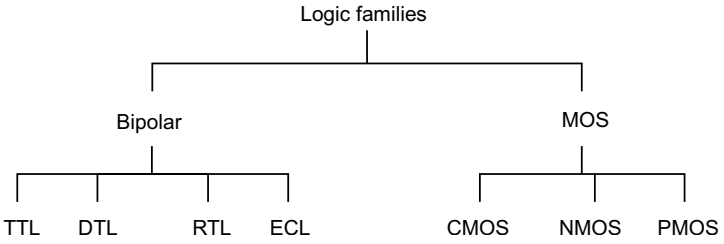


Fig. 3.7.1

Q.2 *Define noise margin.* **Dec.-04; May-07, 08; Dec.-08**

Ans. :

- The voltage difference between the lowest possible HIGH output, $V_{OH\,(min)}$ and the minimum input voltage, $V_{IH\,(min)}$ required for a HIGH input is called **high-state noise margin.**
- The voltage difference between the largest possible low output, $V_{OL\,(max)}$ and the maximum voltage, $V_{IL\,(max)}$ required for a LOW input is called **low-state noise margin.**
- The noise margin allows the digital circuit to function properly if noise voltages are within the noise margin.

Q.3 *What is fan-out of a gate?* **Dec.-05, 08**

Ans. : It is the maximum number of inputs of the same family that the gate can drive maintaining its output levels within the specified limits.

For example : If an inverter's output can drive maximum 10 inputs of any logic gates from same family, the fan-out of the inverter is 10.

Q.4 *What is fan-in of a gate?*

Ans. : Fan-in of a gate is said to be the number of inputs in a digital logic gate.

For example : 2 input NOR has fan-in of 2.

Q.5 *Define power dissipation and propagation delay.* **May-05**

Ans. : Power dissipation is the product of I_{CC} and V_{CC}, where I_{CC} is the average supply current and V_{CC} is the supply voltage. It is given as :

Power dissipation $P_D = I_{CC} \times V_{CC}$.

The propagation delay of a gate is basically the time interval between the application of an input pulse and the occurrence of the resulting output pulse.

Q.6 *Why does the propagation delay occur in logic circuits ?* **Dec.-06**

Ans. : Propagation delay occur in logic families due to following reasons :

1. The intrinsic delay of the gate itself, also known as **internal delay** (t_{Po}).
2. The delay introduced by fan-outs of a logic gate. It is denoted as t_{PL}.

Hence, for a fan-out of N of a gate, the **propagation delay** (t_p) is approximately,

$$t_p = t_{Po} + t_{PL}$$

3. While logic gate levels are cascaded then resulting propagation delay will be addition of propagation delays of each level.

Q.7 ***What do you mean by logic levels ? Explain positive logic and negative logic systems.***

Ans. : The two voltage levels used to represent two logic states in digital logic are called **logic levels.**

The system in which the higher of the two voltage levels represent a 1 and the lower voltage represents a 0 is called a **positive logic system**.

Positive Logic HIGH = 1 (i.e. +5 V)

LOW = 0 (i.e. 0 V)

The system in which the lower of the two voltage levels represent a 1 and the higher of the voltage represents a 0 is called a **negative logic system.**

Negative Logic HIGH = 0 (i.e. +5 V)

LOW = 1 (i.e. 0 V)

Q.8 ***What are the advantages of digital integrated circuits ?***

Ans. : Advantages of digital integrated circuits are :

1. The traditional method of designing combinational circuit is effective for small circuit. But time consuming and less reliable for complex circuits digital ICs take care of this problem.
2. Digital ICs reduce system cost.
3. They are smaller in size.
4. Less power is required for the system.
5. Lesser number of external wire connections which improves reliability of the system.
6. Depending upon circuit complexity, any of the 5 levels of integration can be chosen.

Q.9 ***Define speed power product.***

Ans. : Speed power product is the product of propagation delay and average power dissipation. It is a means for measuring and comparing the overall performance of an IC family. It is given as :

$$\text{Speed power product (J)} = \text{Propogation delay}(\text{seconds}) \times \text{Power dissipation}(\text{J / sec})$$

Q.10 ***Mention the important characteristics of digital IC's.***

Ans. : The important characteristics of digital IC's are :

1. Fan out
2. Power dissipation
3. Propagation delay

4. Noise margin
5. Fan in
6. Operating temperature
7. Power supply requirements.

Q.11 ***What is operating temperature ?***

Ans. : All the gates or semiconductor devices are temperature sensitive in nature. The temperature in which the performance of the IC is effective is called operating temperature. Operating temperature of the IC vary from 0 ºC to 70 ºC.

Q.12 ***When does the noise margin allow digital circuits to function properly.***

Ans. : When noise voltages are within the limits of (High state noise margin and low state noise margin for a particular logic family; the digital circuits are allowed to function properly.

Q.13 ***Define rise time and fall time.***

Ans. :

- **Rise time** is the time required to switch from 10 % of voltage level to 90 of voltage level.
- **Fall time** is the time required to switch from 90 % of the voltage level to 10 % of voltage level.

Q.14 ***Explain output switching times using waveform.***

Ans. : The finite time taken by a circuit to change output levels is called output switching time. There are 2 terms of great significance in output switching times. They are : t_{LH} (rise time) and t_{HL} (fall time). The minimum amount of time needed for a gate to switch from 0 to 1 and back is given by

$$t_{min} = t_{LH} + t_{HL}$$

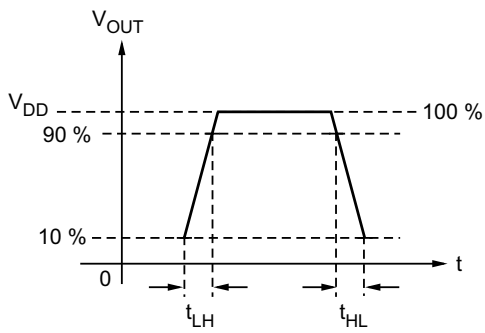


Fig. 3.7.2 Output switching times

Q.15 *Differentiate : Source and sink current.* **May-07**

Ans. :

Sr. No.	Current sinking	Current sourcing
1.	When the output of the logic circuit is low, the current flows from load into the pull down transistor of the totem-pole. This is known as **current sinking.**	When the output of the logic circuit is high, the current flows from pull up transistor of the totem-pole to the load. This is known as **current sourcing.**
2.	During current sinking base-emitter diode of the load is forward biased. Hence sinking current is in milliamperes.	During current sourcing only leakage current flows hence sourcing current is in microamperes.

Q.16 *What are the types of TTL logic?*

Ans. : 1. Open collector output 2. Totem-pole output 3. Tri-state output.

Q.17 *List the different versions of TTL.*

Ans. :

1. TTL (Std.TTL) 2. LTTL (Low Power TTL)
3. HTTL (High Speed TTL) 4. STTL (Schottky TTL)
5. LSTTL (Low power Schottky TTL)

Q.18 *Distinguish between 7400 series and 5400 series.* **May-04**

Ans. :

Supply voltage	For 74 series - (4.75 to 5.25) V For 54 series - (4.5 to 5.5) V
Temperature range	For 74 series - (0 °C to 70 °C) For 54 series - (– 55 °C to 125 °C)

Q.19 *What is meant by tristate capability?* **May-07**

Ans. : Tristate capability is the capability of output level to take three stages (output levels) as HIGH, LOW and High-impedance. For example, tristate TTL.

Q.20 *Explain open collector output.*

Ans. : When the collector terminal of a transistor is kept open without any pull up transistor the arrangement is called open collector output. The output is taken directly from the open collector terminal of a transistor at the output. But, a gate with open collector will not work properly until an external resistor is connected.

When Q_4 is ON, output is low and when Q_4 is OFF, output is tied to V_{CC} through an external pull up resistor.

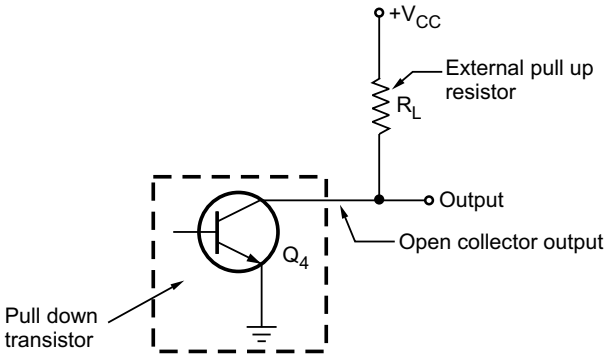


Fig. 3.7.3 Open collector output with pull-up resistor

Q.21 ***Compare the totem-pole and open-collector outputs.***

Ans. : Refer Table 3.3.3.

Q.22 ***Describe the characteristics of TTL family.***

Ans. : Refer Table 3.3.5 standard TTL characteristics.

Q.23 ***How Schottky transistors are formed and state its use?***

Ans. : A Schottky diode is formed by the combination of metal and semiconductor. The presence of schottky diode between the base and the collector prevents the transistor from going into saturation. The resulting transistor is called Schottky transistor.

The use of Schottky transistor in TTL decreases the propagation delay without a sacrifice of power dissipation.

Q.24 ***Why totem pole outputs cannot be connected together?***

Ans. : Totem pole outputs cannot be connected together because such a connection might produce excessive current and may result in damage to the devices.

Q.25 ***What happens to output when a tristate circuit is selected for high impedance.***

Ans. : Output is disconnected from rest of the circuits by internal circuitry.

Q.26 ***State advantages and disadvantages of totem-pole output.***

Ans. : Advantages of totem-pole output are :

1. External pull up resistor is not required.
2. Operating speed is high.

Disadvantages of totem-pole output :

1. Output of two gates cannot be tied together.

Q.27 *Explain the wired-AND connection.*

Ans. : When the open collector outputs of two or more gates can be connected together, the connection is called a wired AND. In wired-AND connection, the output is high only when all switches are open i.e. when output of each stage is high.

Hence, the output is equivalent to the logical AND operation of the logic function performed by the gates.

Q.28 *Give any two applications of open collector logic.* **May-07**

Ans. :

1. It is used when we require to connect outputs of two or more gates.
2. It is used when sourcing current requirements is larger than the standard totem-pole output.

Q.29 *State advantages and disadvantages of TTL.*

Ans. : Refer section 3.3.9.

Q.30 *What is the effect of increasing the supply voltage on the propagation delay of the CMOS gates ?* **May-09**

Ans. : Increasing the supply voltage reduce the propagation delay of the CMOS gates.

❑❑❑

4 Combinational Circuits

Syllabus

Design procedure - Half adder - Full adder - Half subtractor - Full subtractor - Parallel binary adder, Parallel binary subtractor - Fast adder - Carry look ahead adder - Serial adder/subtractor - BCD adder - Binary multiplier - Binary divider - Multiplexer/demultiplexer - Decoder - Encoder - Parity checker - Parity generators - Code converters - Magnitude comparator.

Contents

4.1 Introduction
4.2 Design Procedure
4.3 Design of Adders . . . . . ***May-03, 06, 11, Dec.-03, 07, 10,*** · · · · · · · · *Marks 8*
4.4 Design of Subtractor. . . . . ***May-03, 10, 11, Dec.-10*** · · · · · · · · · · · · · · *Marks 7*
4.5 Parallel Adder
4.6 Parallel Subtractor
4.7 Parallel Adder/Subtractor . . . . . ***May-03, Dec.-07, 12,*** · · · · · · · · · · · · · · · · · *Marks 8*
4.8 Look Ahead-Carry Generator (Fast Adder) . . ***Dec.-10,*** · *Marks 2*
4.9 Four-Bit Parallel Adder (7483/74283)
4.10 Serial Adder/Subtractor . . . . . ***Dec.-11, 12,*** · *Marks 2*
4.11 BCD Adder . . . . . ***May-12, Dec.-12,*** · *Marks 16*
4.12 Binary Multiplier
4.13 Binary Divider . . . . . ***May-07, 09, Dec.-07, 09,*** · · · · · · · · · · · · · · · *Marks 8*
4.14 Multiplexer ***May-07, 08, 09, 11,***
. . . . . ***Dec.-03, 06, 08, 09, 11,*** · · · · · · · · · · · · · · · *Marks 10*
4.15 Demultiplexer . . . . . ***May-04*** · *Marks 2*
4.16 Decoder . . . . . ***Dec.-06, May-07*** · *Marks 2*
4.17 Encoders
4.18 Parity Generators and Checkers . . . . . ***Dec.-08, 12, May-10*** · · · · · · · · · · · · · · · · · *Marks 6*
4.19 Code Converters ***Dec.-03, 04, 09, 12,***
. . . . . ***May-05, 09, 10*** · *Marks 16*
4.20 Design of Comparator . . . . . ***May-08, 11, 12,*** · *Marks 16*
4.21 Two Marks Questions with Answers

4.1 Introduction

When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called **combinational logic circuit**. In combinational logic circuit, the output variables are at all times dependent on the combination of input variables.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from the input variables and generate output signals. This process transforms binary information from the given input data to the required output data. Fig. 4.1.1 shows the block diagram of a combinational circuit. As shown in Fig. 4.1.1, the combinational circuit accepts n-input binary variables and generates output variables depending on the logical combination of gates.

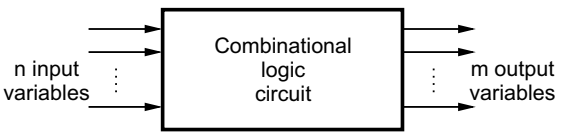


Fig. 4.1.1 Block diagram of a combinational circuit

Review Question

1. *What is combinational logic circuit ?*

4.2 Design Procedure

The design of combinational circuits starts from the outline of the problem statement and ends in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be easily obtained. The design procedure of the combinational circuit involves following steps :

1. The problem definition.
2. The determination of number of available input variables and required output variables.
3. Assigning letter symbols to input and output variables.
4. The derivation of truth table indicating the relationships between input and output variables.
5. Obtain simplified Boolean expression for each output.
6. Obtain the logic diagram.

Example for Understanding

Example 4.2.1 *Design a combination logic circuit with three input variables that will produce a logic 1 output when more than one input variables are logic 1.*

Solution :

Step 1 : Derive the truth table for given statement.

Given problem specifies that there are three input variables and one output variable. We assign A, B and C letter symbols to three input variables and assign Y letter symbol to one output variable. The relationship between input variables and output variable can be tabulated as shown in truth Table 4.2.1.

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 4.2.1 Truth table

Step 2 : Obtain simplified Boolean expression.

Now we obtain the simplified Boolean expression for output variable Y using K-map simplification.

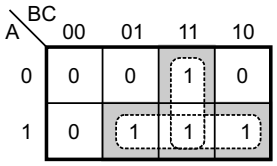


Fig. 4.2.1 K-map simplification

$\therefore \quad Y = AC + BC + AB$

Step 3 : Draw logic diagram.

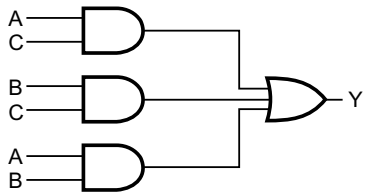


Fig. 4.2.2 Logic diagram

In this chapter we are going to study various combinational circuits using above illustrated design method.

Examples with Solutions

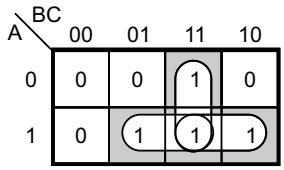
Example 4.2.2 *A majority gate is a digital circuit whose output is equal to 1 if the majority of inputs are 1's. The output is 0 otherwise. Using a truth table, find the Boolean function implemented by a 3-input majority gate. Simplify the function and implement with gates.*

Dec.-07, Marks 8

Solution :

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 4.2.2 Truth table



Y = BC + AB + AC

Fig. 4.2.3 K-map simplification

Implementation with 3-input gates

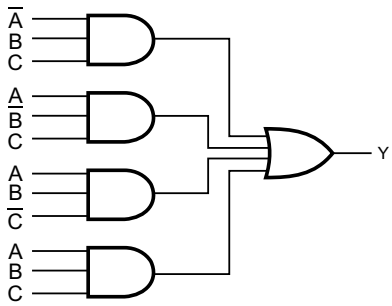


Fig. 4.2.4

Implementation after simplification

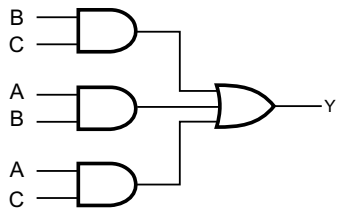


Fig. 4.2.5

Example 4.2.3 *The inputs to a circuit are the 4 bits of the binary number $D_3D_2D_1D_0$. The circuit produces a 1 if and only if all of the following conditions hold.*

1) MSB is '1' or any of the other bits are a '0'.

2) D_2 is a 1 or any of the other bits are a '0'. 3) Any of the 4 bits are a 0.

Obtain a minimal expression for the output.

Dec.-08, Marks 8

Solution : The truth table for the given problem is as shown in Table 4.2.3.

Inputs				Output
D_3	D_2	D_1	D_0	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 4.2.3

K-map Simplification

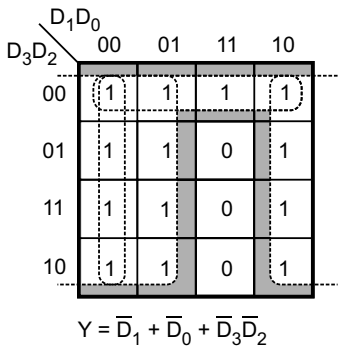


Fig. 4.2.6

Review Question

1. *Explain the design procedure for combinational circuits.*

4.3 Design of Adders

May-03, 06, 11, Dec.-03, 07, 10

Digital computers perform various arithmetic operations. The most basic operation, no doubt, is the addition of two binary digits. This simple addition consists of four possible elementary operations, namely,

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10_2$

The first three operations produce a sum whose length is one digit, but when the last operation is performed sum is two digits. The higher significant bit of this result is called a **carry**, and lower significant bit is called **sum**. The logic circuit which performs this operation is called a **half-adder**. The circuit which performs addition of three bits (two significant bits and a previous carry) is a **full-adder**.

4.3.1 Half Adder

The half-adder operation needs two binary inputs : augend and addend bits; and two binary outputs : sum and carry. The truth table shown in Table 4.3.1 gives the relation between input and output variables for half-adder operation.

Inputs		Outputs	
A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 4.3.1 Truth table for half-adder

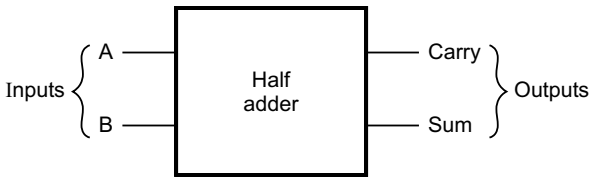


Fig. 4.3.1 Block schematic of half-adder

K-map simplification for carry and sum

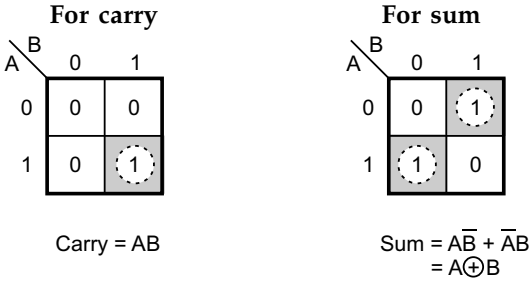


Fig. 4.3.2 Maps for half-adder

Logic diagram

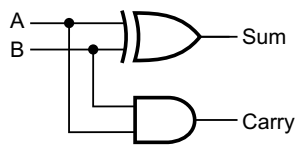


Fig. 4.3.3 Logic diagram for half-adder

Limitations of Half-Adder :

In multidigit addition we have to add two bits along with the carry of previous digit addition. Effectively such addition requires addition of three bits. This is not possible with half-adder. Hence half-adders are not used in practice.

Example 4.3.1 *Draw half adder using NAND gates.*

Solution : For half adder :

$$\text{Sum} = A\overline{B} + \overline{A}B \qquad C_{out} = AB$$

$$= \overline{\overline{A\overline{B}} + \overline{\overline{A}B}} \qquad = \overline{\overline{AB}}$$

$$= \overline{\overline{A\overline{B}} \cdot \overline{\overline{A}B}}$$

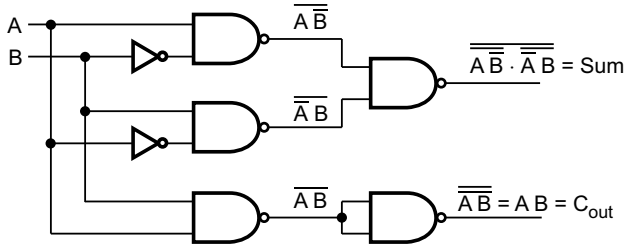


Fig. 4.3.4

4.3.2 Full Adder

A full-adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the input variables, denoted by A and B, represent the two significant bits to be added. The third input C_{in}, represents the carry from the previous lower significant position. The truth table for full-adder is shown in Table 4.3.2.

Inputs			Outputs	
A	**B**	**C_{in}**	**Carry**	**Sum**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.3.2 Truth table for full-adder

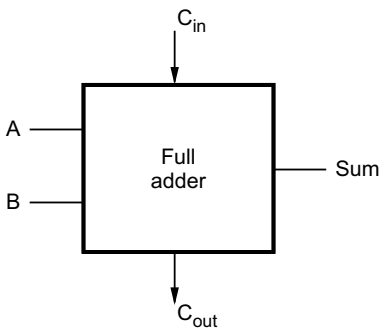
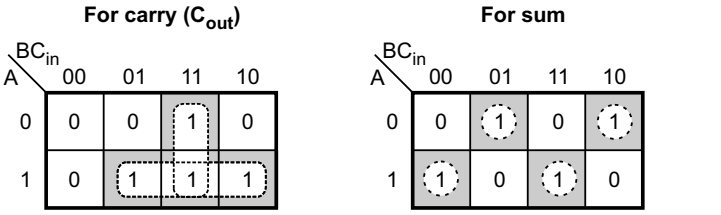


Fig. 4.3.5 Block schematic of full-adder

K-map simplification for carry and sum



$C_{out} = AB + A C_{in} + B C_{in}$

$Sum = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$

Fig. 4.3.6 Maps for full-adder

Logic diagram

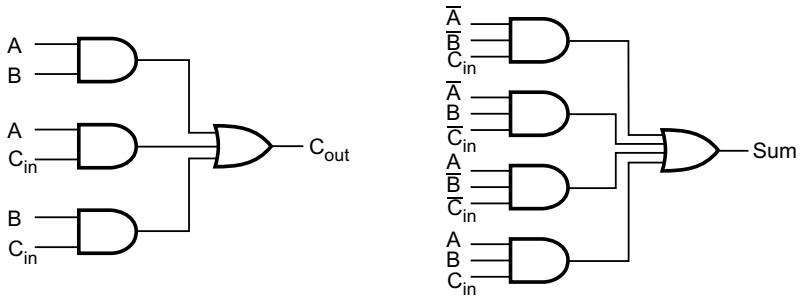


Fig. 4.3.7 Sum of product implementation of full-adder

The Boolean function for sum can be further simplified as follows :

$$\begin{aligned} Sum &= \overline{A}\ \overline{B}\, C_{in} + \overline{A}\, B\, \overline{C}_{in} + A\, \overline{B}\ \overline{C}_{in} + A\, B\, C_{in} \\ &= C_{in}\ (\overline{A}\ \overline{B} + AB) + \overline{C}_{in}\ (\overline{A}\, B + A\, \overline{B}) \\ &= C_{in}\, (A \odot B) + \overline{C}_{in}\ (A \oplus B) \\ &= C_{in}\, (\overline{A \oplus B}) + \overline{C}_{in}\ (A \oplus B) \\ &= C_{in} \oplus (A \oplus B) \end{aligned}$$

With this simplified Boolean function circuit for full-adder can be implemented as shown in the Fig. 4.3.8.

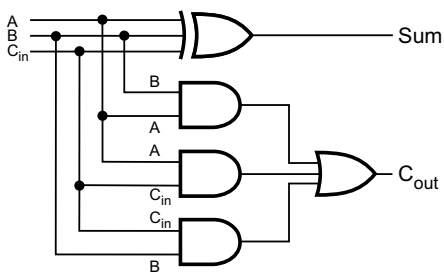


Fig. 4.3.8 Implementation of full-adder

Full adder using two half adders

A full-adder can also be implemented with two half-adders and one OR gate, as shown in the Fig. 4.3.9. The sum output from the second half-adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

$$C_{out} = AB + A\,C_{in} + B\,C_{in} = AB + A\,C_{in}\,(B + \overline{B}) + B\,C_{in}\,(A + \overline{A})$$

$$= AB + ABC_{in} + A\,\overline{B}\,C_{in} + ABC_{in} + \overline{A}\,BC_{in} = AB\,(1 + C_{in} + C_{in}) + A\,\overline{B}C_{in} + \overline{A}\,BC_{in}$$

$$= AB + A\overline{B}C_{in} + \overline{A}BC_{in} = AB + C_{in}\,(A\overline{B} + \overline{A}B) = AB + C_{in}\,(A \oplus B)$$

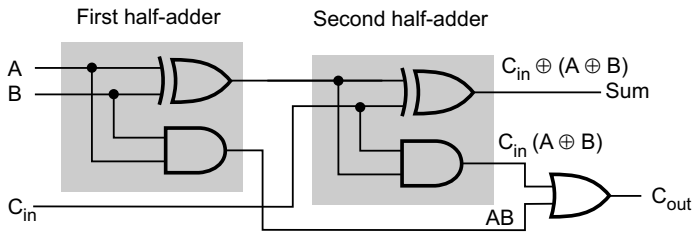


Fig. 4.3.9 Implementation of a full-adder with two half-adders and an OR gate

Review Questions

1. *Design a full adder* **May-03, Dec.-03, Marks 7**
2. *Define half adder and full adder.*
3. *Draw a block diagram of half adder. Write truth table. Draw logic diagram.* **May-06, Marks 2**
4. *Write a truth table for half adder, reduce the equation using K-map and design half adder using logic gates.*
5. *Define full adder. Draw logic circuit and truth table of full adder.* **Dec.-10, Marks 6**
6. *Implement full adder using two half adders.* **Dec.-07, May-11, Marks 8**

4.4 Design of Subtractor

May-03, 10, 11, Dec.-10

The subtraction consists of four possible elementary operations, namely,

$0 - 0 = 0$

$0 - 1 = 1$ with 1 borrow

$1 - 0 = 1$

$1 - 1 = 0$

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of second operation the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed. Just as there are half and full-adders, there are **half** and **full-subtractors.**

4.4.1 Half Subtractor

A half-subtractor is a combinational circuit that subtracts two-bits and produces their difference. It also has an output to specify if a 1 has been borrowed. Let us designate minuend bit as A and the subtrahend bit as B. The result of operation A – B for all possible values of A and B is tabulated in Table 4.4.1.

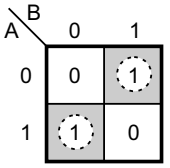
As shown in the Table 4.4.1, half-subtractor has two input variables and two output variables. The Boolean expression for the outputs of half-subtractor can be determined as follows.

Inputs		Outputs	
A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table 4.4.1 Truth table for half-subtractor

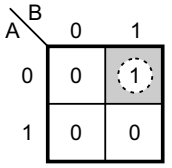
K-map simplification for half-subtractor

For difference



Difference = $A\overline{B} + \overline{A}B$
= $A \oplus B$

For borrow



Borrow = $\overline{A}B$

Fig. 4.4.1

Limitations of half-subtractor :

In multidigit subtraction, we have to subtract two bits along with the borrow of the previous digit subtraction. Effectively such subtraction requires subtraction of three bits. This is not possible with half-subtractor.

Logic diagram

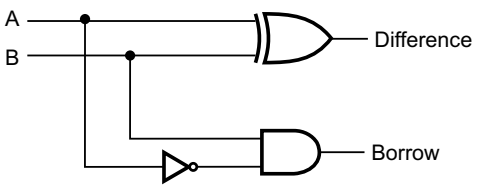


Fig. 4.4.2 Implementation of half-subtractor

Example 4.4.1 *Draw half subtractor using NAND gates.*

Solution : For half subtractor :

$$\text{Difference} = A\overline{B} + \overline{A}B = \overline{\overline{A\overline{B} + \overline{A}B}} = \overline{\overline{A\overline{B}} \cdot \overline{\overline{A}B}}$$

$$\text{Borrow} = \overline{A}B = \overline{\overline{A}B}$$

Implementation :

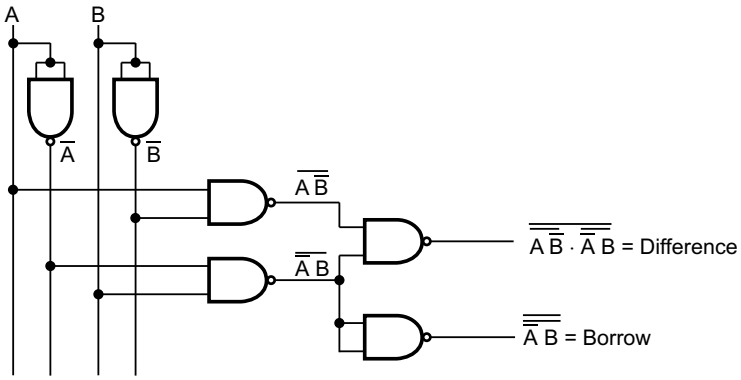


Fig. 4.4.3

4.4.2 Full-Subtractor

A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account borrow of the lower significant stage. This circuit has three inputs and two outputs. The three inputs are A, B and B_{in}, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, D and B_{out}, represent the difference and output borrow, respectively. The Table 4.4.2 shows the truth table for full-subtractor.

Inputs			Outputs	
A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 4.4.2 Truth table for full-subtractor

K-map simplification of D and B_{out}

For D

A \ BB_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$D = \overline{A}\overline{B}B_{in} + \overline{A}B\overline{B}_{in} + A\overline{B}\overline{B}_{in} + ABB_{in}$

For B_{out}

A \ BB_{in}	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$B_{out} = \overline{A}B_{in} + \overline{A}B + BB_{in}$

Fig. 4.4.4 Maps for full-subtractor

Logic diagram

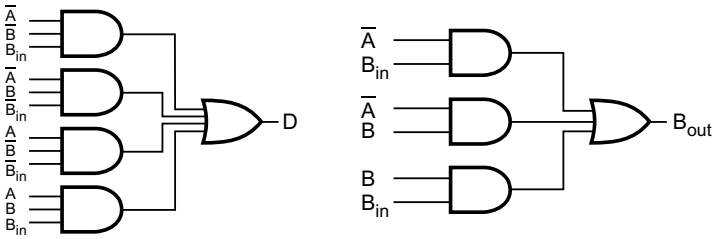


Fig. 4.4.5 Sum of product implementation of full-subtractor

The Boolean function for D (difference) can be further simplified as follows :

$$D = \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B\,\overline{B_{in}} + A\,\overline{B}\,\overline{B_{in}} + A\,B\,B_{in}$$

$$= B_{in}\,(\overline{A}\,\overline{B} + AB) + \overline{B_{in}}\,(\overline{A}\,B + A\,\overline{B})$$

$$= B_{in}\,(A \odot B) + \overline{B_{in}}\,(A \oplus B)$$

$$= B_{in}\,\overline{(A \oplus B)} + \overline{B_{in}}\,(A \oplus B)$$

$$= B_{in} \oplus (A \oplus B)$$

With this simplified Boolean function circuit for full-subtractor can be implemented as shown in the Fig. 4.4.6.

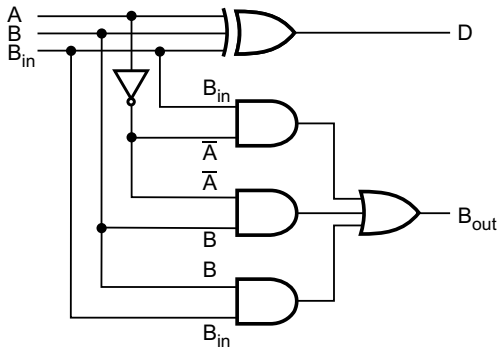


Fig. 4.4.6 Implementation of full-subtractor

A full subtractor can also be implemented with two half-subtractors and one OR gate, as shown in the Fig. 4.4.7. The difference output from the second half-subtractor is the exclusive-OR of B_{in} and the output of the first half-subtractor, which is same as difference output of full-subtractor.

The borrow output for circuit shown in Fig. 4.4.6 can be given as,

$$B_{out} = \overline{A}\,B_{in} + \overline{A}\,B + B\,B_{in}$$

$$= \overline{A}\,B_{in}\,(B + \overline{B}) + \overline{A}\,B + B\,B_{in}\,(A + \overline{A})$$

$$= \overline{A}\,B\,B_{in} + \overline{A}\,\overline{B}\,B_{in} + \overline{A}\,B + AB\,B_{in} + \overline{A}\,B\,B_{in}$$

$$= \overline{A}\,B\,(B_{in} + 1 + B_{in}) + \overline{A}\,\overline{B}\,B_{in} + AB\,B_{in}$$

$$= \overline{A}\,B + \overline{A}\,\overline{B}\,B_{in} + AB\,B_{in}$$

$$= \overline{A}\,B + B_{in}\,(\overline{A}\,\overline{B} + AB)$$

$$= \overline{A}\,B + B_{in}\,\overline{(A \oplus B)}$$

This Boolean function is same as borrow out of the full-subtractor. Therefore, we can implement full-subtractor using two half-subtractors and OR gate.

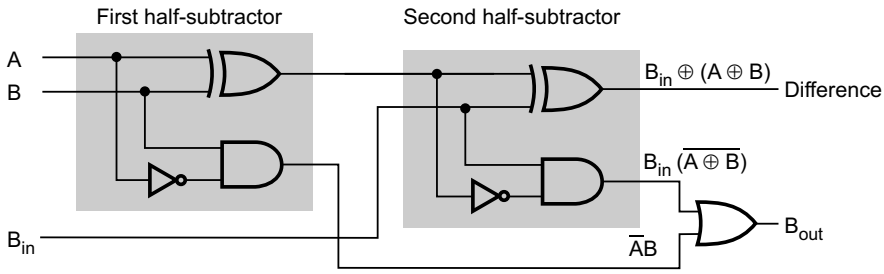


Fig. 4.4.7 Implementation of a full-subtractor with two half-subtractors and an OR gate

Review Questions

1. *Define half subtractor and full subtractor.*
2. *Design a half-subtractor combinational circuit to produce the outputs. Difference and borrow.* **Dec.-10, Marks 2**
3. *Explain the operation of a half subtractor with the help of logic diagram and truth table.*
4. *Write the logic expressions for the difference and borrow of a half subtractor.* **May-11, Marks 2**
5. *Realize full-subtractor using K-map.*
6. *Write an expression for borrow and difference in a full subtractor circuit.* **May-10, Marks 2**
7. *Design full-subtractor circuit and draw necessary truth tables.* **May-03, Marks 7**
8. *Explain how full subtractor can be designed by using two half subtractor circuits. Draw the circuit diagram.*

4.5 Parallel Adder

We have seen, a single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders must be employed. A n-bit, parallel adder can be constructed using number of full adder circuits connected in parallel. Fig. 4.5.1 shows the block diagram of n-bit parallel adder using number of full-adder circuits connected in cascade, i.e. the carry output of each adder is connected to the carry input of the next higher-order adder.

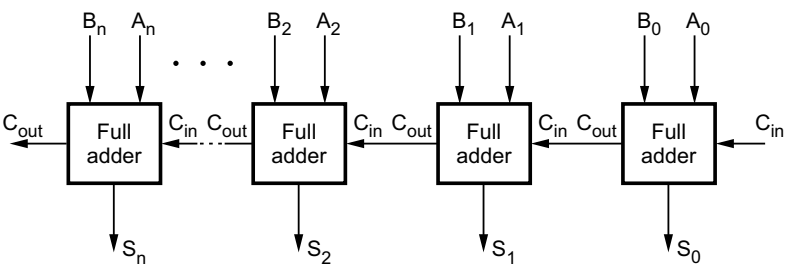


Fig. 4.5.1 Block diagram of n-bit parallel adder

It should be noted that either a half-adder can be used for the least significant position or the carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

Example 4.5.1 *Design a 4-bit parallel adder using full-adders.*

Solution : Fig. 4.5.2 shows the block diagram for 4-bit adder. Here, for least significant position, carry input of full-adder is made 0.

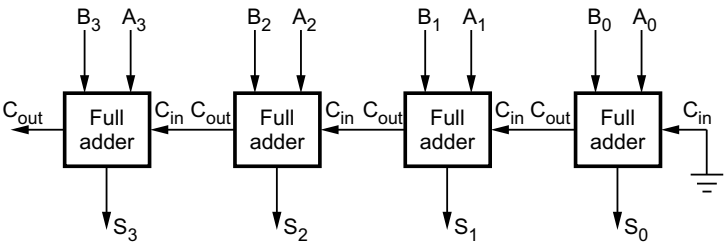


Fig. 4.5.2 Block diagram of 4-bit full-adder

Review Questions

1. *What is parallel adder ?*
2. *Draw and explain the working of 4-bit parallel adder.*

4.6 Parallel Subtractor

The subtraction of binary numbers can be done most conveniently by means of complements. Remember that the subtraction A-B can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry, as shown in the Fig. 4.6.1.

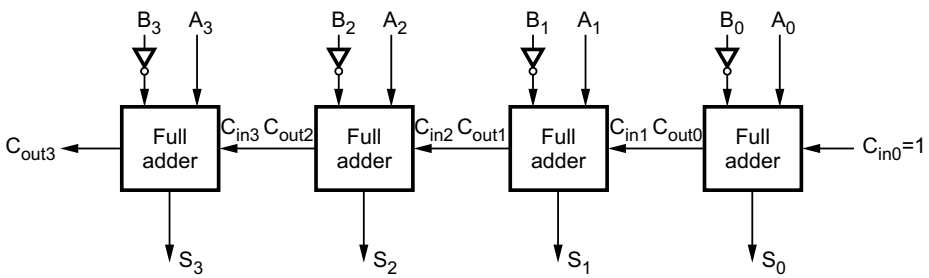


Fig. 4.6.1 4-bit parallel subtractor

Review Question

1. *Draw and explain 4-bit parallel subtractor.*

4.7 Parallel Adder/Subtractor

May-03, Dec.-07, 12

The addition and subtraction operations can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder, as shown in Fig. 4.7.1. The mode input M controls the operation of the circuit. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor. Each exclusive - OR gate receives input M and one of the inputs of B. When M = 0, we have $B \oplus 0 = B$. The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When M = 1, we have $B \oplus 1 = \overline{B}$ and $C_{in\ 0} = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2's complement of B, i.e. A – B.

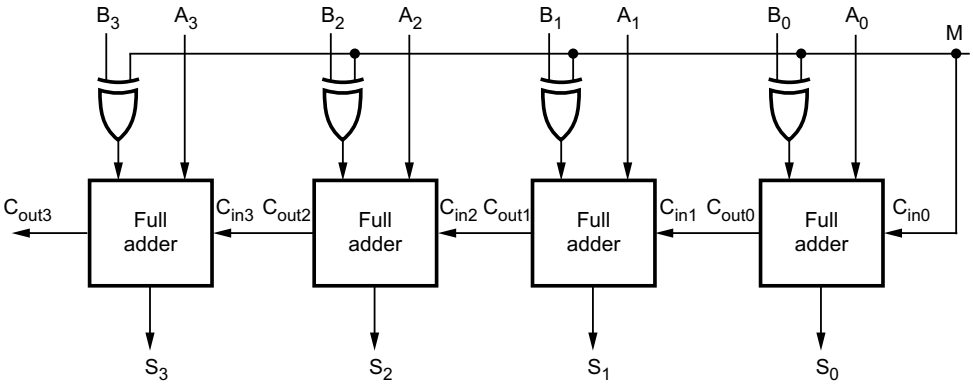


Fig. 4.7.1 4-bit adder-subtractor

The parallel adder is ripple carry adder in which the carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Therefore, the sum and carry outputs of any stage cannot be produced until the input carry occurs; this leads to a time delay in the addition process. This delay is known as **carry propagation delay**.

One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition.** This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated.

Review Questions

1. *What is the drawback in binary parallel adder? How can it be rectified ?* **Dec.-07, Marks 3**
2. *Draw the block diagram of a 2's complement adder/subtractor.* **May-03, Marks 2**
3. *Design a 4 bit parallel adder/substractor and draw the logic diagram.* **Dec.-12, Marks 8**

4.8 Look Ahead-Carry Generator (Fast Adder) Dec.-10

The parallel adder discussed in the last paragraph is ripple carry type in which the carry output of each full-adder stage is connected to the carry input of the next higher-order stage. Therefore, the sum and carry outputs of any stage cannot be produced until the input carry occurs; this leads to a time delay in the addition process. This delay is known as *carry propagation delay*, which can be best explained by considering the following addition.

$$\begin{array}{rr} & 0\,1\,0\,1 \\ + & 0\,0\,1\,1 \\ \hline & 1\,0\,0\,0 \end{array}$$

Addition of the LSB position produces a carry into the second position. This carry, when added to the bits of the second position (stage), produces a carry into the third position. The latter carry, when added to the bits of the third position, produces a carry into the last position. The key thing to notice in this example is that the sum bit generated in the last position (MSB) depends on the carry that was generated by the addition in the previous positions. This means that, adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full-adder is considered to have a propagation delay of 30 ns, then S_3 will not reach its correct value until 90 ns after LSB carry is generated. Therefore, total time required to perform addition is 90 + 30 = 120 ns.

Obviously, this situation becomes much worse if we extend the adder circuit to add a greater number of bits. If the adder were handling 16-bit numbers, the carry propagation delay could be 480 ns.

One method of speeding up this process by eliminating inter stage carry delay is called **look ahead-carry addition.** This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions : carry generate and carry propagate.

Consider the circuit of the full-adder shown in Fig. 4.8.1. Here, we define two functions : carry generate and carry propagate.

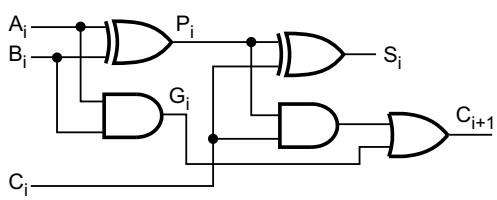


Fig. 4.8.1 Full-adder circuit

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output sum and carry can be expressed as

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i + P_i C_i$$

G_i is called a carry generate and it produces on carry when both A_i and B_i are one, regardless of the input carry. P_i is called a carry propagate because it is term associated with the propagation of the carry from C_i to C_{i+1}.

Now the Boolean function for the carry output of each stage can be written as follows.

$$C_2 = G_1 + P_1C_1$$

$$C_3 = G_2 + P_2C_2 = G_2 + P_2 (G_1 + P_1C_1)$$

$$= G_2 + P_2G_1 + P_2 P_1C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1C_1)$$

$$= G_3 + P_3G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$

From the above Boolean function it can be seen that C_4 does not have to wait for C_3 and C_2 to propagate; in fact C_4 is propagated at the same time as C_2 and C_3.

The Boolean functions for each output carry are expressed in sum-of product form, thus they can be implemented using AND-OR logic or NAND-NAND logic. Fig. 4.8.2 shows implementation of Boolean functions for C_2, C_3 and C_4 using AND-OR logic.

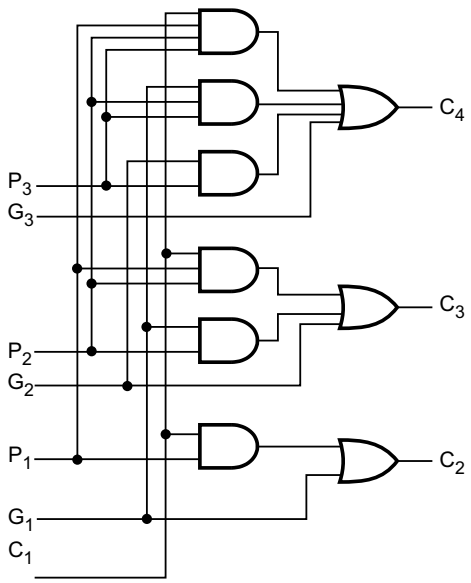


Fig. 4.8.2 Logic diagram of a look ahead carry generator

Using a look ahead carry generator we can easily construct a 4-bit parallel adder with a look ahead carry scheme. Fig. 4.8.3 shows a 4-bit parallel adder with a look ahead carry scheme. As shown in the Fig. 4.8.3, each sum output requires two exclusive-OR gates. The output of first exclusive-OR gate generates P_i , and the AND gate generates G_i. The carries are generated using look ahead carry generator and applied as inputs to the second exclusive-OR gate. Other inputs to exclusive-OR gate is P_i . Thus second exclusive-OR gate generates sum outputs. Each

output each generated after a delay of two levels of gate. Thus outputs S_2 through S_4 have equal propagation delay times.

IC 74182 is a look ahead carry generator. Fig. 4.8.3 shows pin diagram and logic symbol for IC 74182. As shown in the logic symbol, the 74182 carry look ahead generator accepts up to four pairs of active low carry propagate $(\overline{P}_0, \overline{P}_1, \overline{P}_2, \overline{P}_3)$ and carry generate $(\overline{G}_0, \overline{G}_1, \overline{G}_2, \overline{G}_3)$ signals and an active high carry input (C_n) and provides anticipated active high carries $(C_{n+x}, C_{n+y}, C_{n+z})$ across four groups of binary adders. The 74182 also has active low carry propagate $(\overline{P})$ and carry generate $(\overline{G})$ outputs which may be used for further levels of look ahead.

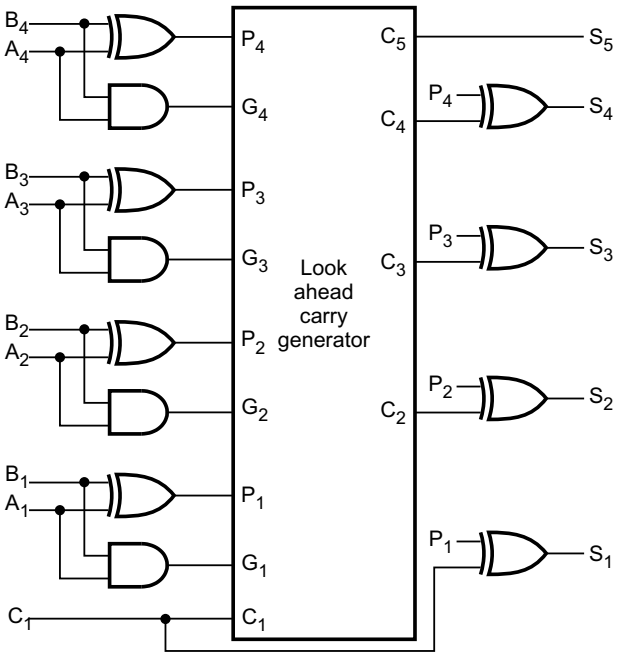
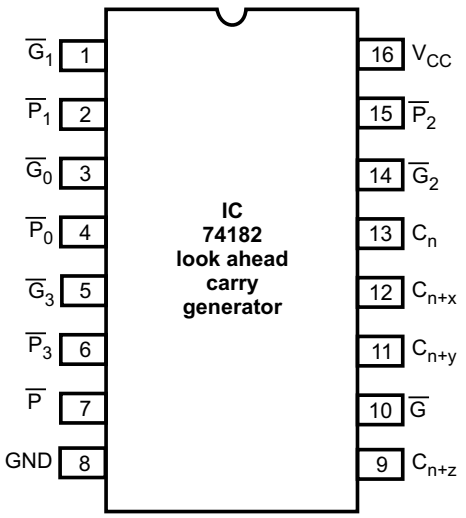
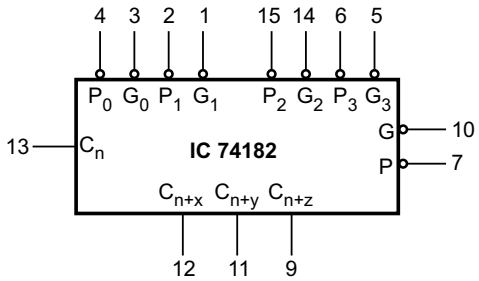


Fig. 4.8.3 4-bit parallel adder with look ahead carry generator



(a) Pin diagram



(b) Logic symbol

Fig. 4.8.4

The logic equations provided at the outputs of 74182 are :

$$C_{n+x} = G_0 + P_0 C_n$$

$$C_{n+y} = G_1 + P_1 G_0 + P_1 P_0 C_n$$

$$C_{n+z} = G_2 + P_2 G_1 + P_2 P_1 G_0$$

$$\overline{G} = \overline{G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0}$$

$$\overline{P} = \overline{P_3 P_2 P_1 P_0}$$

Example 4.8.1 *Show the construction of 4-bit parallel adder using IC 74182.*

Solution : Fig. 4.8.5 shows the construction of 4-bit parallel adder using IC 74182. The last carry output can be generated using $\overline{P}$ and $\overline{G}$ outputs of IC 74182, as shown in the Fig. 4.8.5.

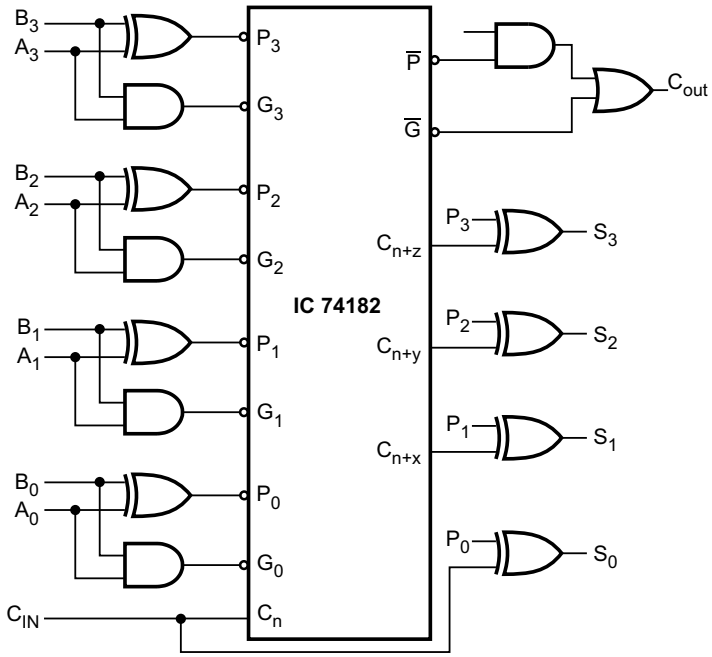


Fig. 4.8.5 4-bit parallel adder using IC 74182

Example 4.8.2 *Construct the look ahead carry generator to accommodate higher word size.*

Dec.-11, Marks 8

Solution : Look ahead carry generators can be cascaded to increase the word size in multiples of 4-bits. Fig. 4.8.6 shows the cascading two look ahead carry generators (IC 74182s) to get word size of 8-bits.

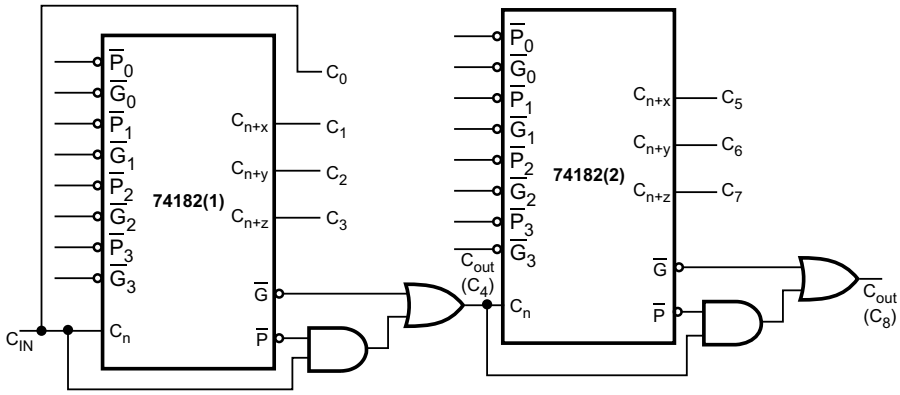


Fig. 4.8.6 Cascading two look ahead carry generators

Review Questions

1. *What do you mean by carry propagation delay ?*
2. *Explain the method used for fast addition (carry look ahead generation).*
3. *Relate carry generate, carry propagate, sum and carry-out of a carry look ahead adder.*

Dec.-10, Marks 2

4.9 Four-Bit Parallel Adder (7483/74283)

Many high-speed adders available in integrated-circuit form utilize the look-ahead carry or a similar technique for reducing overall propagation delays. The most common is a 4-bit parallel adder IC (74LS83/74S283) that contains four inter connected full-adders and the look-ahead carry circuitry needed for high-speed operation. The 7483 and 74283 are a TTL medium scale integrated (MSI) circuit with same pin configuration. Fig. 4.9.1 shows the logic diagram and the functional symbol for the 74LS283 4-bit parallel adder. The inputs to this IC are two 4-bit numbers, $A_3\ A_2\ A_1\ A_0$ and $B_3\ B_2\ B_1\ B_0$, and the carry, C_{in0}, into the LSB position. The outputs are the sum bits $S_3\ S_2\ S_1\ S_0$, and the carry, C_{out3}, output of the MSB position. (See Fig. 4.9.1 (a) show on next page.)

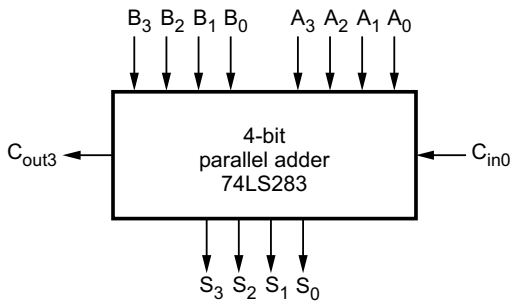


Fig. 4.9.1 (b) Functional symbol for the 74LS283

In the previous section we have used equation $C_{i+1} = G_i + P_i C_i$. Here, the C_{i+1} equation uses the term $P_i G_i$ instead of G_i. This has no effect on the output, since P_i is always 1 when G_i is 1. However, it always the equation to be factorized as follows :

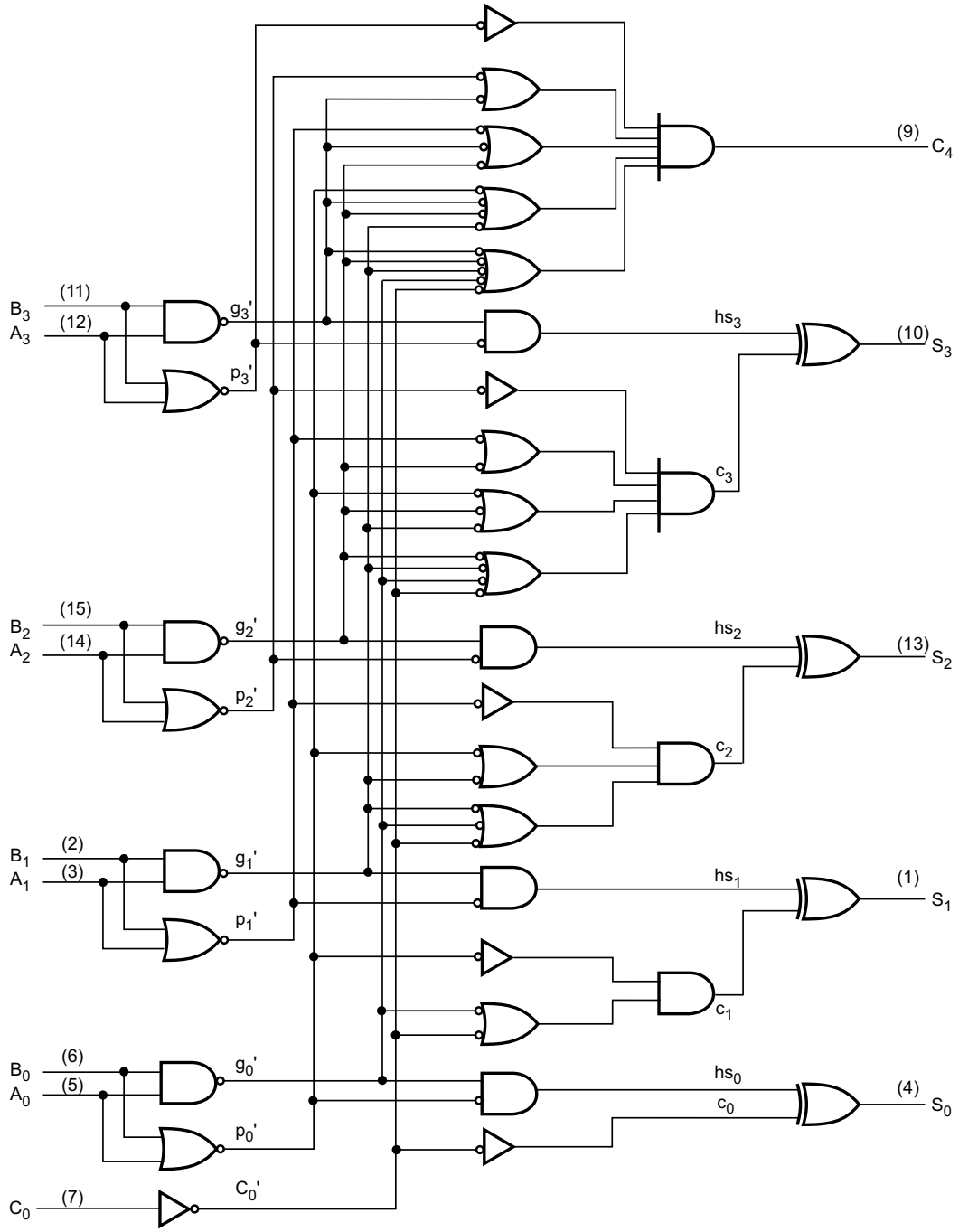


Fig. 4.9.1 (a) Logic diagram for the 74LS283 4-bit binary adder

$$C_{i+1} = P_i \cdot G_i + P_i \cdot C_i$$
$$= P_i \cdot (G_i + C_i)$$

This gives us the following carry equations, which are used by the circuit of 74LS283.

$$C_1 = P_0 \cdot (G_0 + C_0)$$
$$C_2 = P_1 \cdot (G_1 + C_1) = P_1 \cdot \left(G_1 + P_0 \cdot (G_0 + C_0)\right)$$
$$= P_1 \cdot (G_1 + P_0) \cdot (G_1 + G_0 + C_0)$$
$$C_3 = P_2 \cdot (G_2 + C_2) = P_2 \cdot \left(G_2 + P_1 \cdot (G_1 + P_0) \cdot (G_1 + G_0 + C_0)\right)$$
$$= P_2 \cdot (G_2 + P_1) \cdot (G_2 + G_1 + P_0) \cdot (G_2 + G_1 + G_0 + C_0)$$
$$C_4 = P_3 \cdot (G_3 + C_3)$$
$$= P_3 \cdot \left(G_3 + P_2 \cdot (G_2 + P_1) \cdot (G_2 + G_1 + P_0) \cdot (G_2 + G_1 + G_0 + C_0)\right)$$
$$= P_3 \cdot (G_3 + P_2) \cdot (G_3 + G_2 + P_1) \cdot (G_3 + G_2 + G_1 + P_0) \cdot (G_3 + G_2 + G_1 + G_0 + C_0)$$

Two or more parallel adder blocks can be connected (cascaded) to accommodate the addition of larger binary numbers. This is illustrated in example 4.9.1.

Example 4.9.1 *Design an 8-bit adder using two 74LS283s.*

Solution : Fig. 4.9.2 shows how two 74LS283 adders can be connected to add two 8-bit numbers. The adder on the right adds the four least significant bits of the numbers. The C_{out3} output of this adder is connected as the input carry to the first position of the second adder, which adds the four most significant bits of the numbers. The eight sum outputs represent the resultant sum of the two 8-bit numbers. C_{out7} is the carry out of the last position (MSB) of the second adder. It can be used as an overflow bit or as a carry into another adder stage if large binary numbers are to be handled.

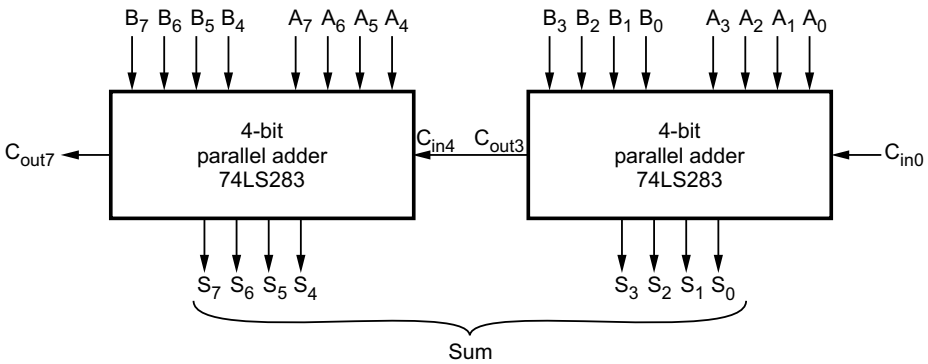


Fig. 4.9.2

Example 4.9.2 *Design a 24-bit group ripple adder using 74X283 ICs.*

Solution :

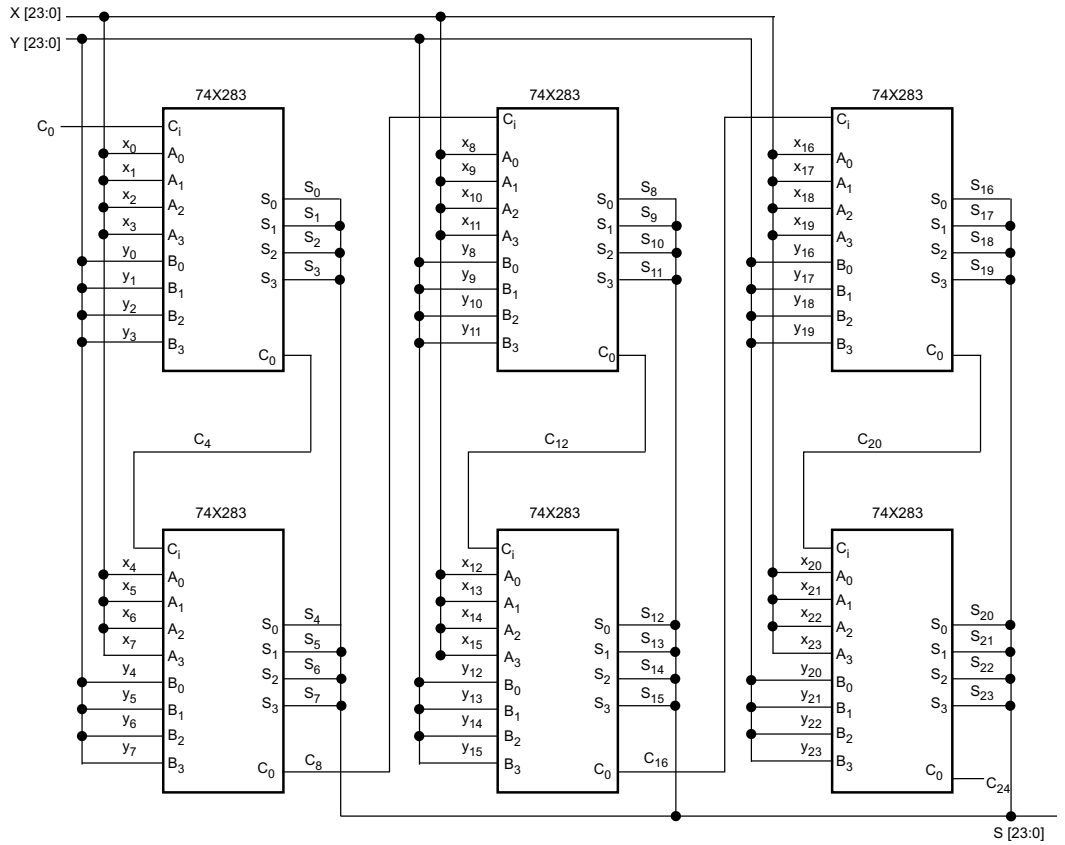


Fig. 4.9.3 A 24-bit group ripple adder using 74X283 ICs

4.10 Serial Adder/Subtractor

Dec.-11, 12

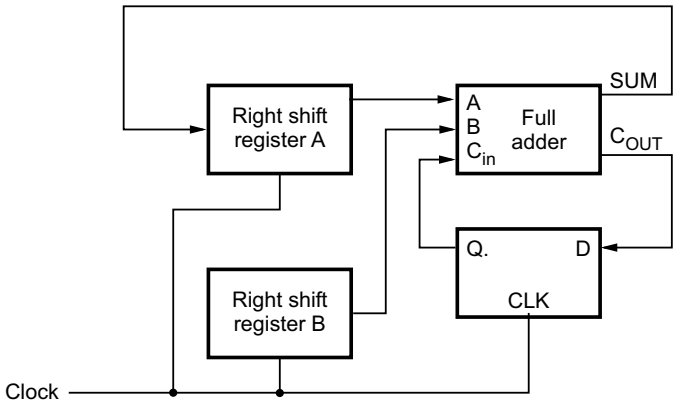


Fig. 4.10.1 Serial adder

The Fig. 4.10.1 shows the logic diagram of serial adder. Using the logic diagram shown in Fig. 4.10.1 we can add numbers stored in the right shift registers A and B, serially. (Readers are suggested to read details of shift register and D-flip-flop from chapter 7 and 5, respectively). The full-adder

is used to perform bit by bit addition and D-flip-flop is used to store the carry output generated after addition. This carry is used as carry input for the next addition. Initially, the D-flip-flop is cleared and addition starts with the least significant bits of both register. After each clock pulse data within the right shift registers are shifted right 1-bit and we get bits from next digit and carry of previous addition as new inputs for the full-adder. The result SUM is stored bit by bit in the register A.

We can implement serial subtractor by replacing full-subtractor instead of full-adder in the Fig. 4.10.1. Here, we will get difference and borrow instead of sum and carry.

Comparison between serial and parallel adder

Sr. No.	Serial adder	Parallel adder
1.	Serial adder uses shift registers.	Parallel adder uses registers with parallel load capacity.
2.	The serial adder requires only one full-adder circuit.	The number of full-adder circuits in the parallel adder equal to the number of bits in the binary numbers.
3.	The serial adder is a sequential circuit.	Excluding the registers, the parallel adder is a purely combinational circuit.
4.	Time required for addition depends on number of bits.	Time required for addition does not depend on number of bits.
5.	It is slower.	It is faster.

Review Questions

1. *Draw and explain serial adder.*
2. *Draw the logic diagram of a serial adder.* **Dec.-12, Marks 2**
3. *Compare the performance of binary serial adder and parallel adders.* **Dec.-11, Marks 2**

4.11 BCD Adder/Subtractor

May-12, Dec.-12

4.11.1 BCD Adder

The digital systems handle the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD digits and produces a sum digit also in BCD. Here, we will see the implementation of addition of BCD numbers.

To implement BCD adder we require :

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 0110_2 in the sum if sum is greater than 9 or carry is 1.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of given truth table.

Y = 1 indicates sum is greater than 9. We can put one more term, C_{out} in the above expression to check whether carry is one. If any one condition is satisfied we add 6 (0110) in the sum.

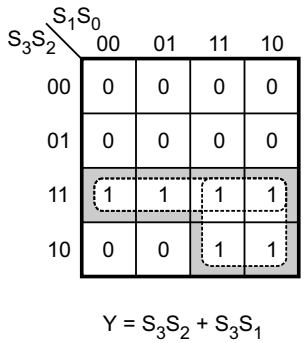
With this design information we can draw the block diagram of BCD adder, as shown in the Fig. 4.11.1 (b).

As shown in the Fig. 4.11.1 (b), the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum.

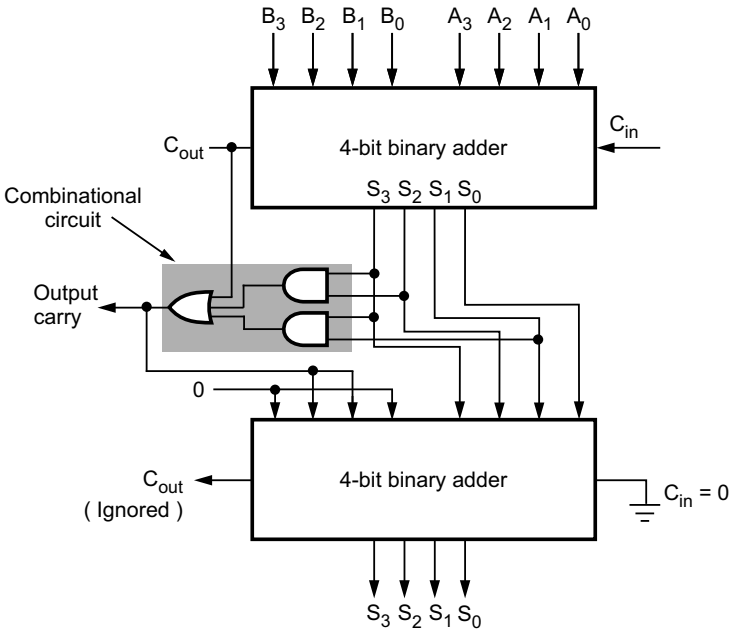
When the output carry is equal to zero (i.e. when sum ≤ 9 and C_{out} = 0) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum > 9 or C_{out} = 1), binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1

Table 4.11.1



(a) K-map simplification



(b) Block diagram of BCD adder

Fig. 4.11.1

output carry generated from the bottom binary adder can be ignored, since it supplies information already available at the output-carry terminal.

Example 4.11.1 *Design an 8-bit BCD adder using 4-bit binary adder.*

Solution : To implement 8-bit BCD adder we have to cascade two 4-bit BCD adders. In cascade connection carry output of the lower position (digit) is connected as a carry input of the higher position (digit). Fig. 4.11.2 shows the block diagram of 8-bit BCD adder.

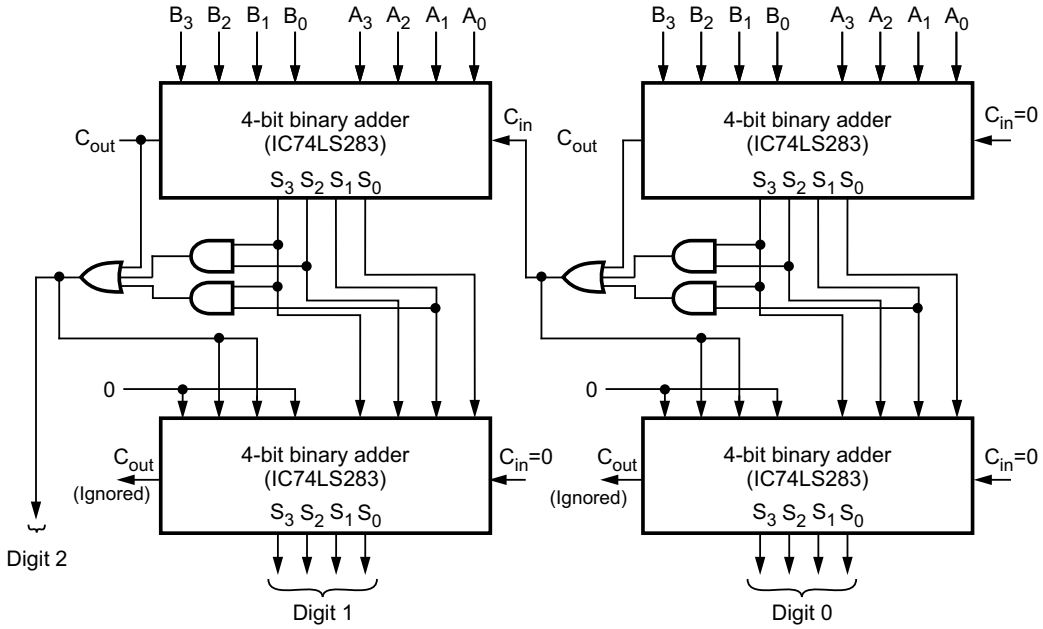


Fig. 4.11.2 8-bit BCD adder using IC 74283

Example 4.11.2 *Design a BCD to Ex-3 code converter using binary parallel adder.*

Dec.-07, Marks 6

Solution :

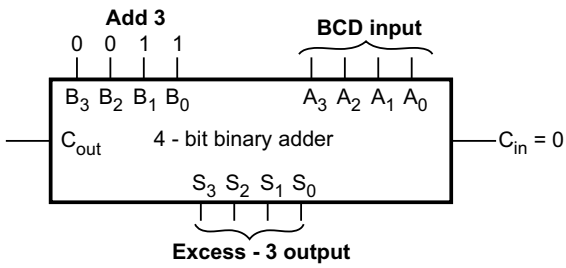


Fig. 4.11.3

4.11.2 BCD Subtractor

Addition of signed BCD numbers can be performed by using 9's or 10's complement methods. A negative BCD number can be expressed by taking the 9's or 10's complement. Let us see 9's and 10's complement numbers and subtraction process using it.

4.11.2.1 9's Complement

The 9's complement of a decimal number is found by subtracting each digit in the number from 9. The 9's complement of each of the decimal digits is as follows :

Digit	9's complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

Table 4.11.2

4.11.2.2 9's Complement Subtraction

In 9's complement subtraction when 9's complement of smaller number is added to the larger number carry is generated. It is necessary to add this carry to the result. (This is called an end-around carry). When larger number is subtracted from smaller one, there is no carry, and the result is in 9's complement form and negative. This is illustrated in following examples :

	Regular Subtraction	9's Complement Subtraction	
(a)	8	8	
	− 2	+ 7	9's complement of 2
	6	(1) 5	
		→ + 1	Add carry to result
		6	
(b)	9	9	
	− 5	+ 4	9' complement of 5
	4	(1) 3	
		→ + 1	Add carry to result
		4	
(c)	4	4	
	− 8	+ 1	9' complement of 8
	− 4	5	9's complement of result
		↓	(No carry indicates that the answer is negative and in complement form).
		− 4	

The 10′s complement of a decimal number is equal to the **9′s complement plus 1**.

4.11.2.3 10's Complement Subtraction

The 10′s complement can be used to perform subtraction by adding the minuend to the 10′s complement of the subtrahend and dropping the carry. This is illustrated in following examples.

	Regular Subtraction	10's Complement Subtraction	
(a)	8	8	
	− 2	+ 8	10's complement of 2
	6	~~1~~ 6	Drop carry
(b)	9	9	
	− 5	+ 5	10's complement of 5
	4	~~1~~ 4	Drop carry
(c)	4	4	
	− 8	+ 2	10's complement of 8
	− 4	6	10's complement of result
		↓	(No carry indicates that the answer is negative and in the 10's complement form)
		− 4	

From the above examples we can summarize steps for 9's complement BCD subtraction as follows :

- Find the 9's complement of a negative number.
- Add two numbers using BCD addition .
- If carry is generated add carry to the result otherwise find the 9's complement of the result.

Fig. 4.11.4 shows the logic diagram of the circuit to implement above mentioned steps to perform BCD subtraction using 9's complement method. As shown in the Fig. 4.11.4, first binary adder finds the 9's complement of the negative number. It does this by inverting each bit of BCD number and adding 10 $(1010)_2$ to it. Let us find the 9's complement of 2.

		0	0	1	0	←	BCD for 2
		1	1	0	1	←	Inverting each bit
	+	1	0	1	0	←	Add 10 $(1010)_2$
Ignore carry	→	1	0	1	1	1	← 9's complement for 2

See Fig. 4.11.4 on next page.

Next two 4-bit binary adders perform the BCD addition. The last adder finds the 9's complement of the result if carry is not generated after BCD addition otherwise it adds carry in the result.

From the above examples we can summarize steps for 10's complement BCD subtraction as follows.

- Find the 10's complement of a negative number.
- Add two numbers using BCD addition.
- If carry is not generated find the 10's complement of the result.

Fig. 4.11.5 shows the logic diagram of the circuit to implement above mentioned steps to perform BCD subtraction using 10's complement method. As shown in the Fig. 4.11.5, first binary adder finds the 10's complement of the negative number (9's complement + 1). Next two 4-bit binary adders perform the BCD addition. Finally, last 4-bit binary adder finds the 10's complement of the number if carry is not generated after BCD addition.

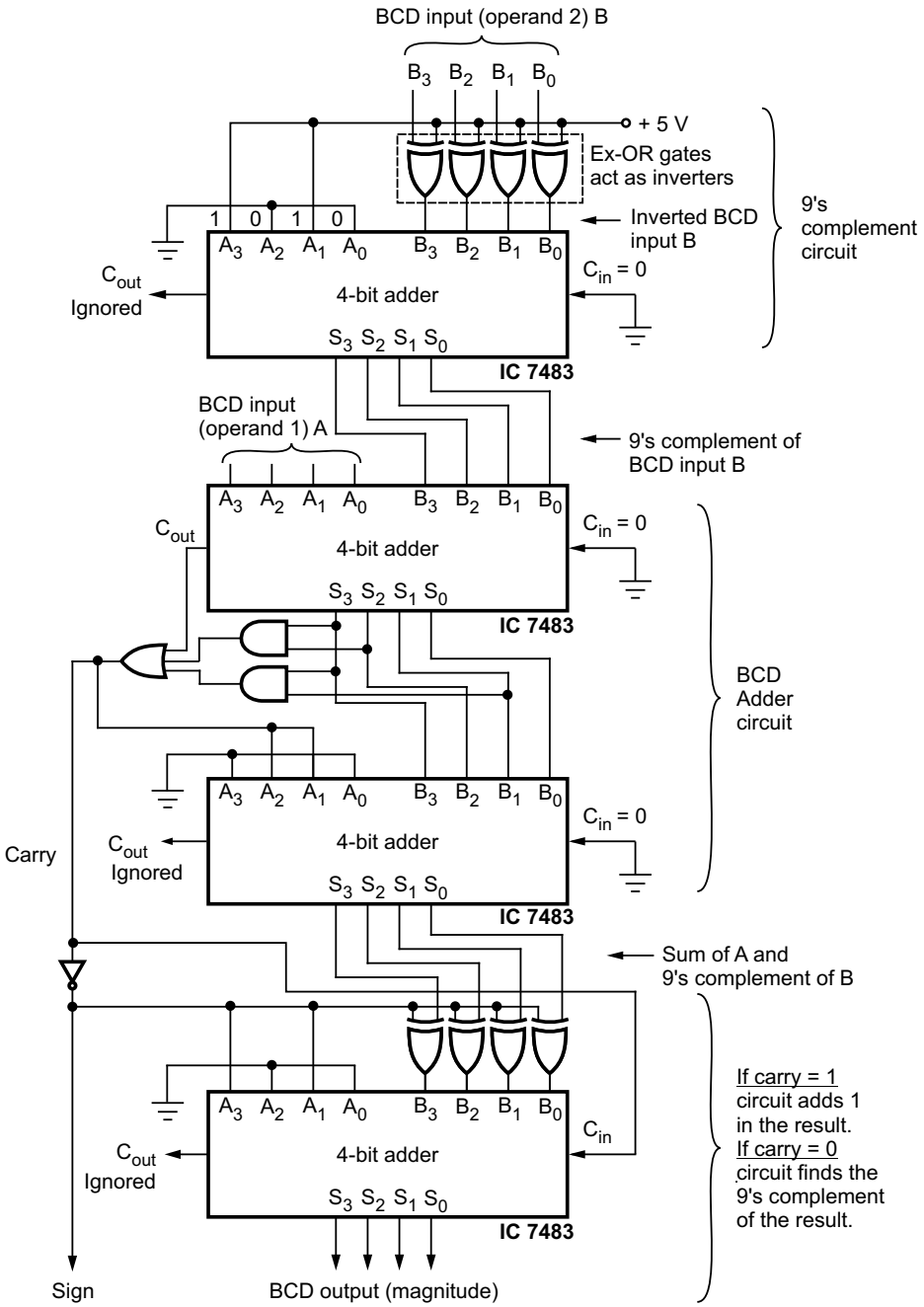


Fig. 4.11.4 4-bit BCD subtractor using 9's complement method

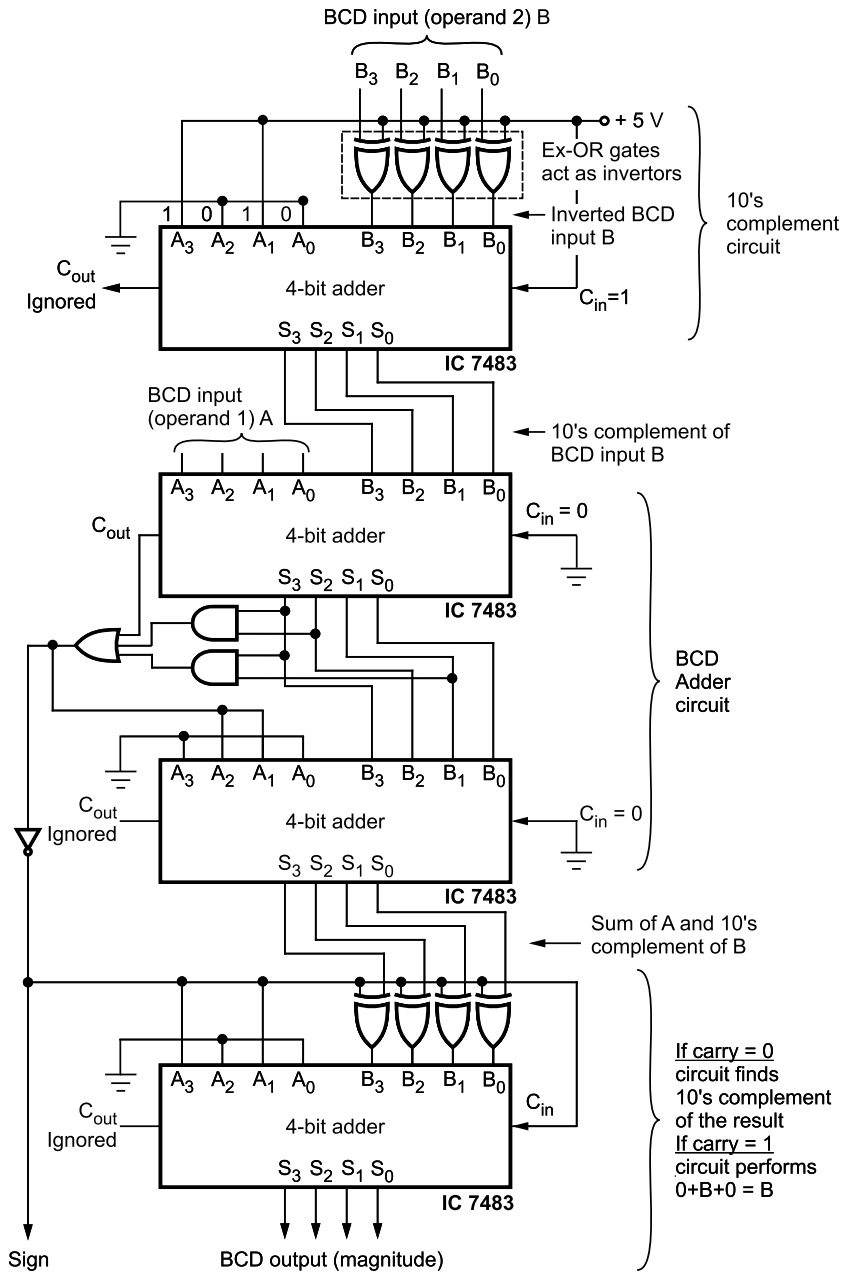

BCD input (operand 2) B
B3 B2 B1 B0
+ 5 V
Ex-OR gates act as invertors
Inverted BCD input B
10's complement circuit
1 0 1 0
A3 A2 A1 A0 B3 B2 B1 B0
Cout Ignored
4-bit adder
S3 S2 S1 S0
Cin=1
IC 7483
BCD input (operand 1) A
10's complement of BCD input B
Cin = 0
BCD Adder circuit
Sum of A and 10's complement of B
If carry = 0 circuit finds 10's complement of the result
If carry = 1 circuit performs 0+B+0 = B
Cin
Sign
BCD output (magnitude)

Fig. 4.11.5 4-bit BCD subtractor using 10's complement method

Review Questions

1. *Design a combinational circuit of 2-Digit BCD Adder.* **May-12, Marks 16**
2. *Draw the logic diagram of a BCD adder and explain its operation.* **Dec.-12, Marks 8**

4.12 Binary Multiplier

The multiplication process for binary numbers is similar to the decimal numbers. Actually binary multiplication is simple than decimal multiplication since it involves only 1s and 0s.

Rules for binary multiplication

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Example 4.12.1 *Multiply 011_2 by 110_2 using binary multiplication method.*

Solution :

$$
\begin{array}{rcccccl}
 & & 0 & 1 & 1 & & \\
 & \times & 1 & 1 & 0 & & \\
\hline
 & & 0 & 0 & 0 & & \\
+ & 0 & 1 & 1 & 0 & & \leftarrow \text{shift left} \\
+ & 0 & 1 & 1 & 0 & 0 & \leftarrow \text{shift left} \\
\hline
 & 1 & 0 & 0 & 1 & 0 &
\end{array}
$$

We have seen how multiplication takes place in binary numbers. It uses n shifts and adds to multiply n-bit binary number. The combination logic circuit implemented to perform such multiplication is called combinational multiplier or **array multiplier.**

Let us generalize the multiplication process for a 2×2 multiplier for two unsigned 2-bit numbers : multiplicand $A = A_1 A_0$ and multiplier B = B1 B0. The Fig. 4.12.1 shows how the multiplication process is carried out.

The multiplication process involves multiplication (product) of 2-bit number and addition of 2-bit number. The

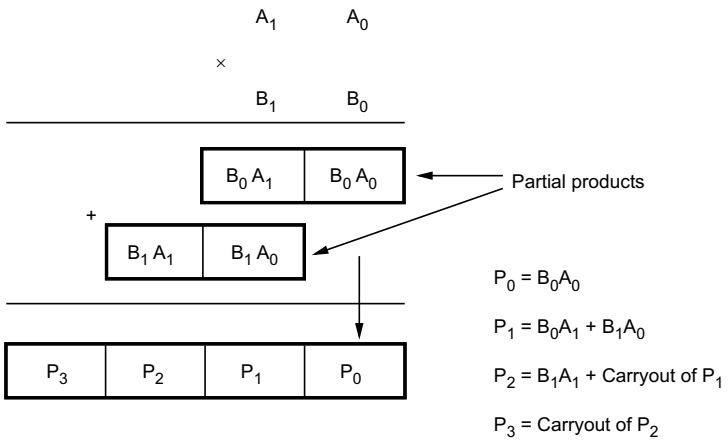


Fig. 4.12.1 Multiplication process

multiplication of 2-bits can be implemented using 2-input AND gate whereas addition of 2-bits can be implemented using half-adder. Such an implementation of 2 × 2 multiplier is shown in the Fig. 4.12.2.

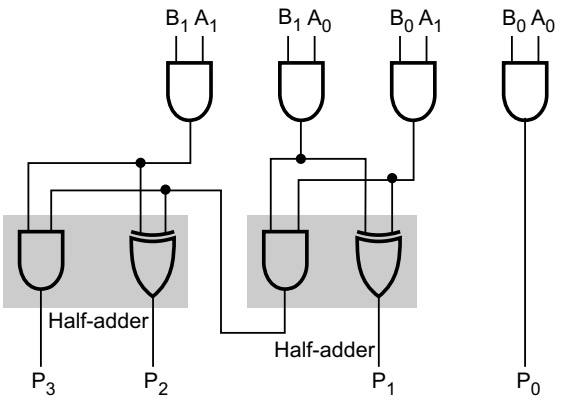


Fig. 4.12.2 2 × 2 bit combinational array multiplier

The Fig. 4.12.3 shows the multiplication process for a 4 × 4 multiplier for two unsigned integers : multiplicand A = A_3 A_2 A_1 A_0 and multiplier B = B_3 B_2 B_1 B_0. As shown in Fig. 4.12.3, each shifted multiplicand which is multiplied by either 0 or 1 depending on the corresponding multiplier bit is called partial product. Each partial product consists of four product component. The product is represented by a product term surrounded by rectangular box. The final 8-bit product is obtained by adding all partial products.

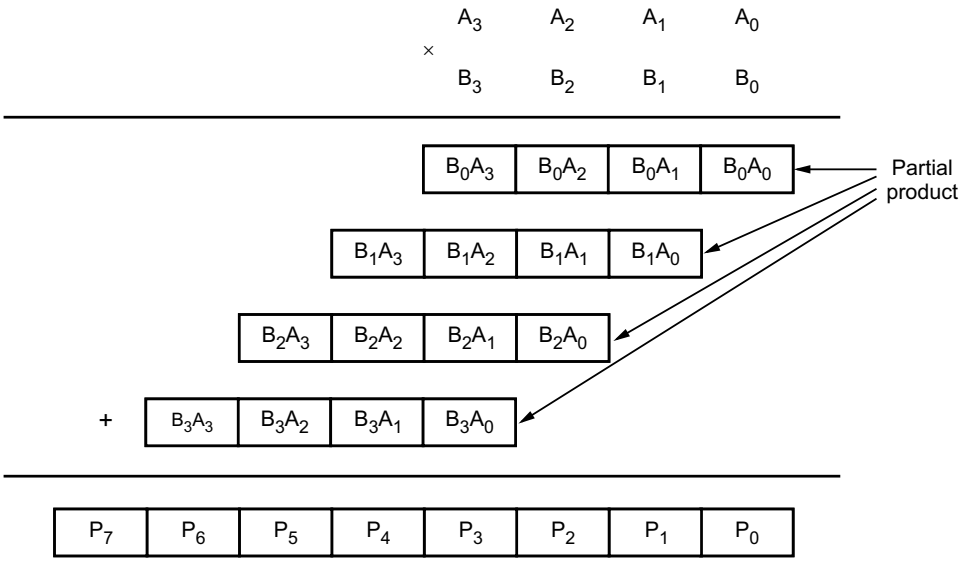


Fig. 4.12.3 4 bit by 4-bit binary multiplication

The first partial product is formed by multiplying B_0 by A_3 A_2 A_1 A_0. The second partial product is formed by multiplying B_1 by A_3 A_2 A_1 A_0. The third partial is formed by multiplying B_2 by A_3 A_2 A_1 A_0 and the fourth partial product is formed by multiplying B_3 by A_3 A_2 A_1 A_0. The multiplication of two bits such as B_0 and A_0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to AND

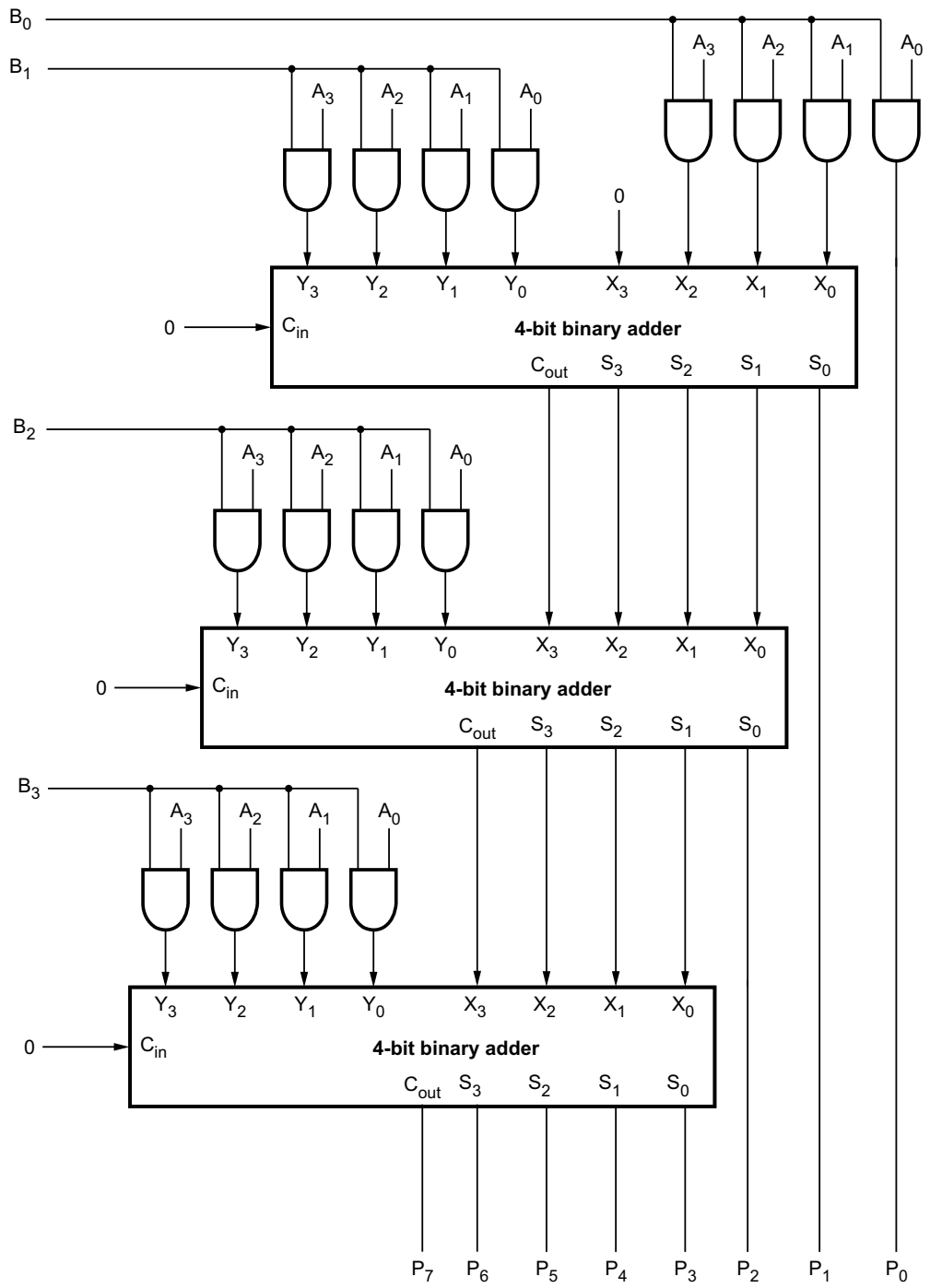


Fig. 4.12.4 4-bit by 4-bit binary multiplier

operation. Therefore, the partial products can be implemented with AND gates as shown in Fig. 4.12.4. The 4-bit partial products are added using 4-bit parallel adder. During addition of first partial product, three most significant bits of it are added to the second partial product. As we take only three bits from first partial product, the fourth (most significant bit) is considered as 0. This is illustrated in Fig. 4.12.1. The three most significant bits and carry out (treated as most significant bit) of first partial sum are then added to the third partial product. Finally the three most significant bits and carry out (treated as most significant bit) of second partial sum are added to the fourth partial product. The carryout and third sum represents the five most significant bits of the product. Least significant bits of first and second partial sum represents P_1 and P_2, respectively, and product B_0A_0 represents P_0. (See Fig. 4.12.4 on previous page).

Example 4.12.2 *Design a multiple circuit to multiply the following binary number $A = A_0A_1A_2$ and $B = B_0B_1B_2B_3$ using required number of binary parallel adders.*

Dec.-11, Marks 10

Solution :

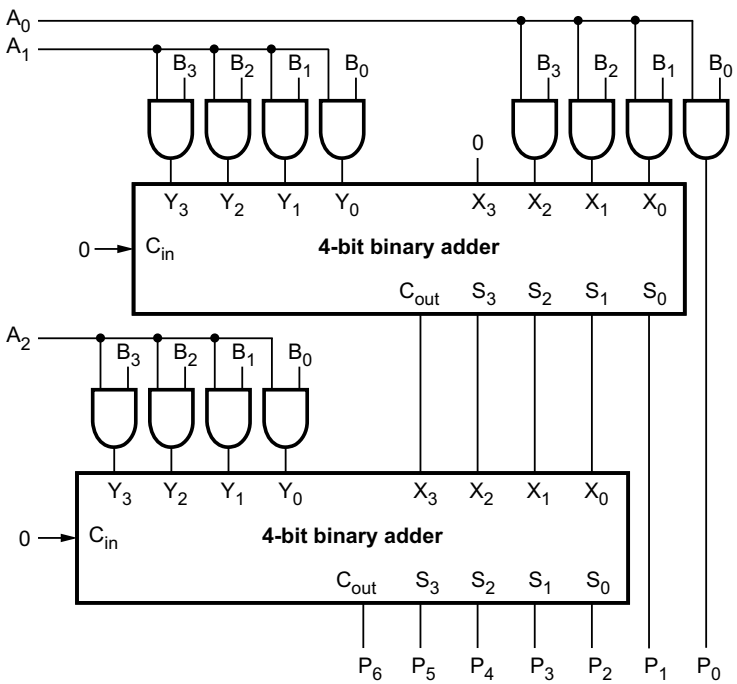


Fig. 4.12.5

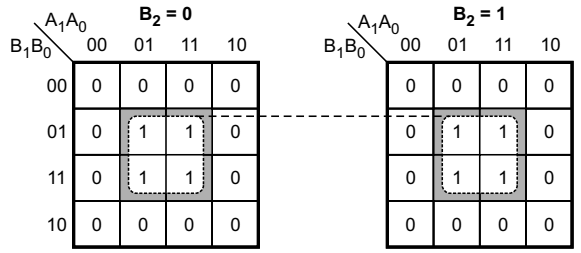
Example 4.12.3 *Design a combinational system that produces the product of 2 binary number* $A = (A_1, A_0)$ *X* $B = (B_2, B_1, B_0)$ **Dec.-10, Marks 10**

Solution :

B_2	B_1	B_0	A_1	A_0	P_4	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	1
0	0	1	1	0	0	0	0	1	0
0	0	1	1	1	0	0	0	1	1
0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	1	1
0	1	1	1	0	0	0	1	1	0
0	1	1	1	1	0	1	0	0	1
1	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	0
1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	1
1	0	1	1	0	0	1	0	1	0
1	0	1	1	1	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	1	1	0
1	1	0	1	0	0	1	1	0	0
1	1	0	1	1	1	0	0	1	0
1	1	1	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1	1	1
1	1	1	1	0	0	1	1	1	0
1	1	1	1	1	1	0	1	0	1

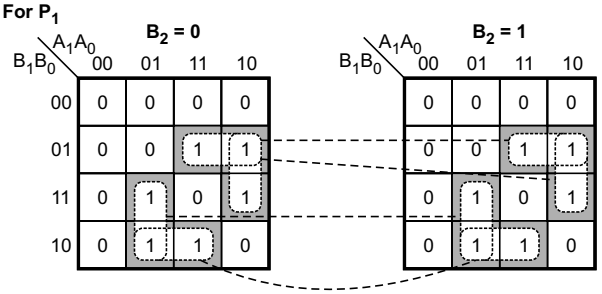
K-map simplification

For P_0



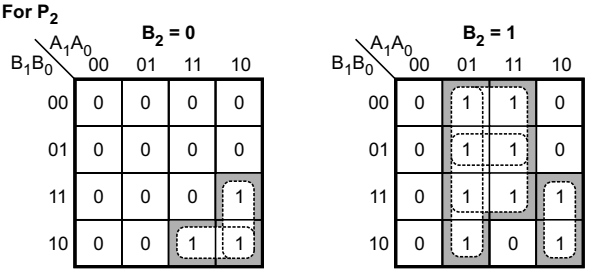
$$P_0 = B_0 A_0$$

For P_1



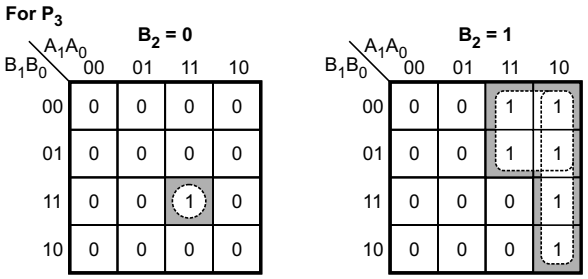
$$P_1 = \overline{B}_1 B_0 A_1 + B_0 A_1 \overline{A}_0 + B_1 \overline{A}_1 A_0 + B_1 \overline{B}_0 A_0$$

For P_2



$$P_2 = \overline{B}_2 B_1 \overline{B}_0 A_1 + B_1 A_1 \overline{A}_0 + B_2 \overline{A}_1 A_0 + B_2 \overline{B}_1 A_0 + B_2 B_0 A_0$$

For P_3



$$P_3 = \overline{B}_2 B_1 B_0 A_1 A_0 + B_2 \overline{B}_1 A_1 + B_2 A_1 \overline{A}_0$$

For P_4

$B_2 = 0$

B_1B_0 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	0	0	0	0

$B_2 = 1$

B_1B_0 \ A_1A_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	1	0
10	0	0	1	0

$P_4 = B_2 B_1 A_1 A_0$

Fig. 4.12.6 (a)

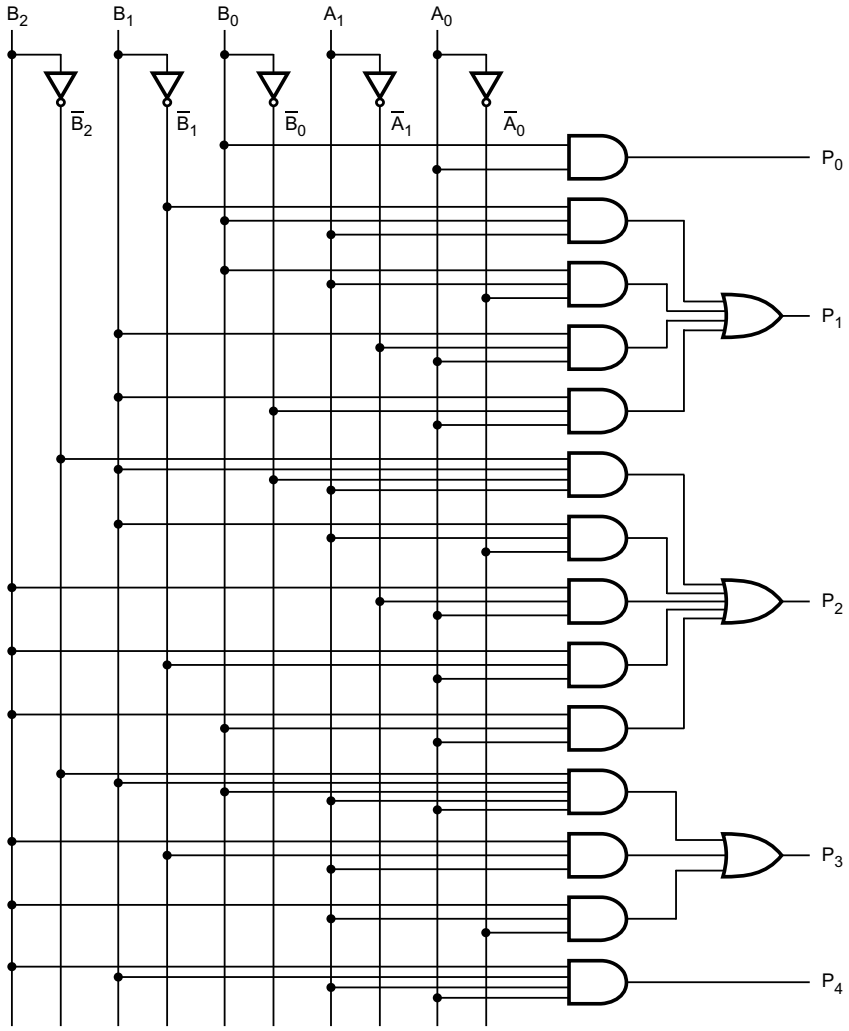


Fig. 4.12.6 (b)

Review Questions

1. *Write a note on binary multiplier.*
2. *Draw and explain the circuit for 4-bit by 4-bit binary multiplier.*

4.13 Binary Divider

May-07, 09, Dec.-07, 09

The division is more complex than multiplication. For simplicity we will see division for positive numbers. Fig. 4.13.1 shows the usual algorithm for dividing positive numbers by hand. It shows examples of decimal division and the binary-coded division of the same value.

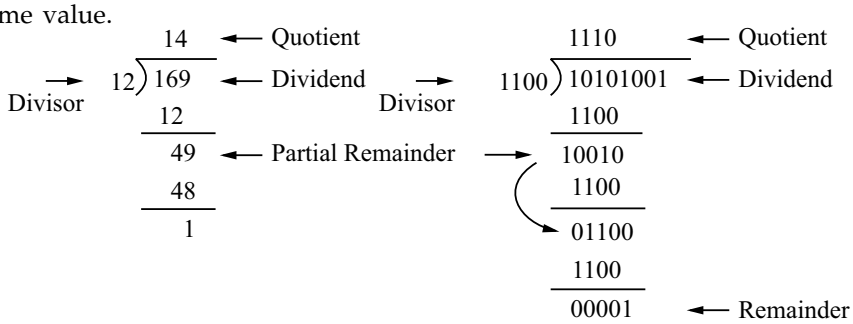


Fig. 4.13.1 Division examples

In both the divisions, division process is same, only in binary division quotient bits are 0 and 1. We will now see the binary division process in detail. First, the bits of the dividend are examined from left to right, until the set of bits examined represents a number greater than or equal to the divisor; this is referred to as the divisor being able to divide the number. Until this condition occurs, 0s are placed in the quotient from left to right. When the condition is satisfied, a 1 is placed in the quotient and the divisor is subtracted from the partial dividend. The result is referred to as a **partial remainder**. From this point onwards, the division process follows repetition of steps. In each repetition cycle, additional bits from the dividend are brought down to the partial remainder until the result is greater than or equal to the divisor, and the divisor is subtracted from the result to produce a new partial remainder. The process continues until all the bits of the dividend are brought down and result is still less than the divisor.

4.13.1 Restoring Division

Fig. 4.13.2 shows the hardware for implementation of division process described in the previous section. It consists of n + 1-bit binary adder, shift, add and subtract control logic and registers A, B, and Q. As shown in the Fig. 4.13.2 divisor and dividend are loaded into register B and register Q, respectively. Register A is initially set to zero. The division operation is then carried out. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. (Fig. 4.13.2 see on next page).

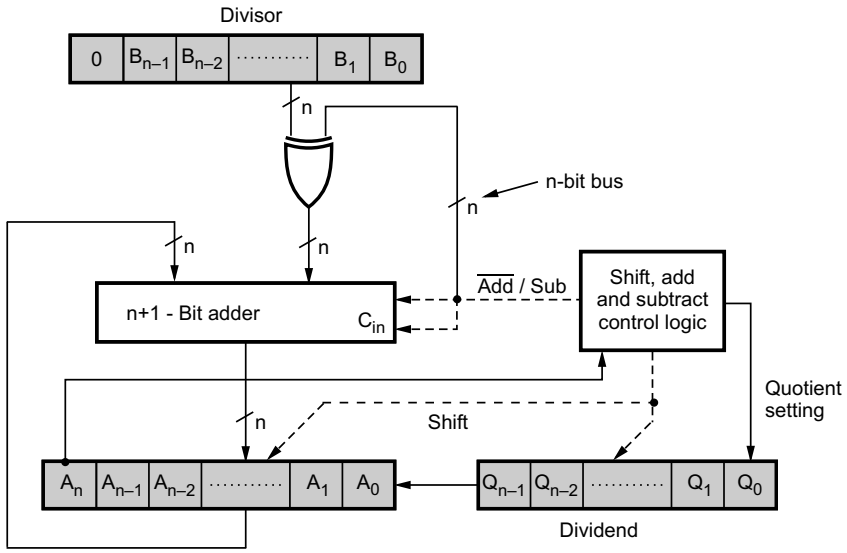


Fig. 4.13.2 Hardware to implement binary division

Division Operation Steps :

1. Shift A and Q left one binary position.
2. Subtract divisor from A and place answer back in A ($A \leftarrow A - B$).
3. If the sign bit of A is 1, set Q_0 to 0 and add divisor back to A (that is, restore A); Otherwise, set Q_0 to 1.
4. Repeat steps 1, 2, and 3 n times.

A flowchart for division operation is as shown in Fig. 4.13.3. (See Fig. 4.13.3 on next page.)

Let us see one example. Consider 4-bit dividend and 2-bit divisor :

Dividend = 1 0 1 0

Divisor = 0 0 1 1

Fig. 4.13.4 shows steps involved in the above binary division. (See Fig. 4.13.3 on next page 4-37)

The division algorithm just discussed needs restoring register A after each unsuccessful subtraction. (Subtraction is said to be unsuccessful if the result is negative). Therefore it is referred to as **restoring division algorithm**. This algorithm is improved, giving *non-restoring division* algorithm. Consider the sequence of operations that takes place after the subtraction operation in the **restoring algorithm**.

If A is positive	If A is negative
Shift left and subtract divisor $\rightarrow 2A - B$	Restore $\rightarrow A + B$ Shift left and subtract divisor $\rightarrow 2 (A + B) - B$ $\equiv 2A + B$

Looking at the above operations we can write following steps for non-restoring algorithm.

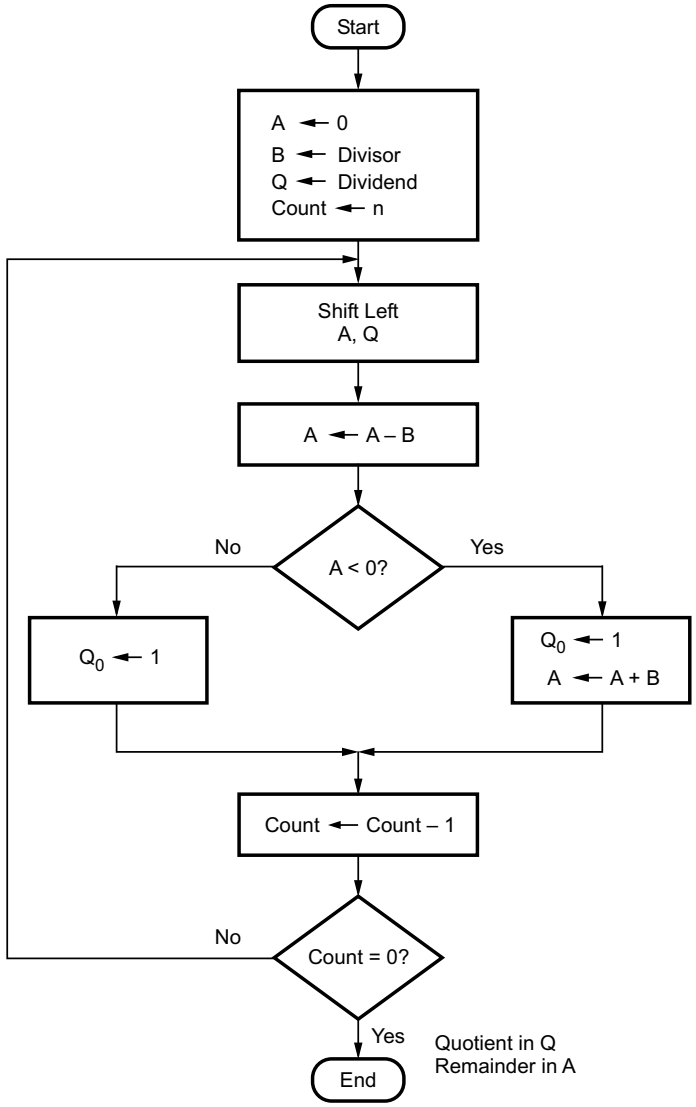


Fig. 4.13.3 Flowchart for restoring division operation

Step 1 : If the sign of A is 0, shift A and Q left one bit position and subtract divisor from A; otherwise, shift A and Q left and add divisor to A. If the sign of A is 0, set Q_0 to 1; otherwise, set Q_0 to 0.

Step 2 : Repeat steps 1 and 2 for n times.

Step 3 : If the sign of A is 1, add divisor to A.

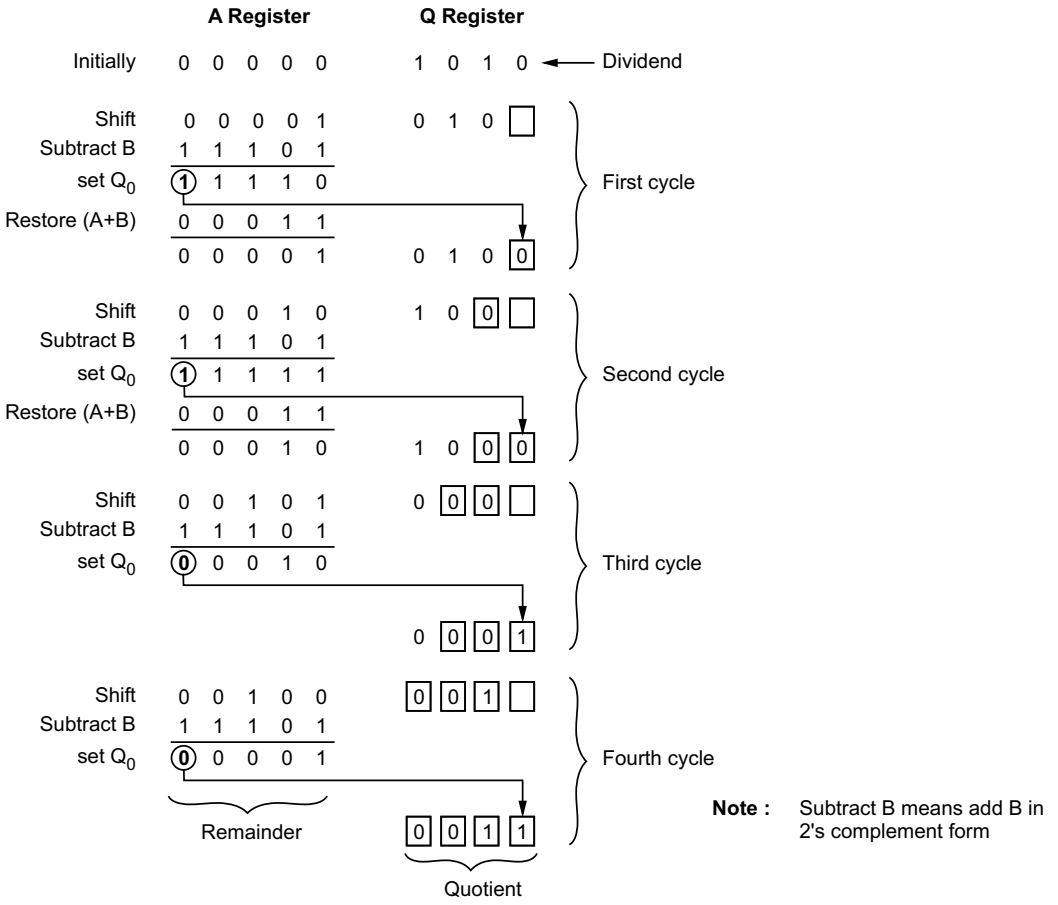


Fig. 4.13.4 A restoring division example

Note : Step 3 is required to leave the proper positive remainder in A at the end of n cycles.

Example 4.13.1 *Perform 1100 ÷ 11 using restoring algorithm .*

Solution : Restoring Division : Q ← Dividend (1100), B ← Divisor (11).

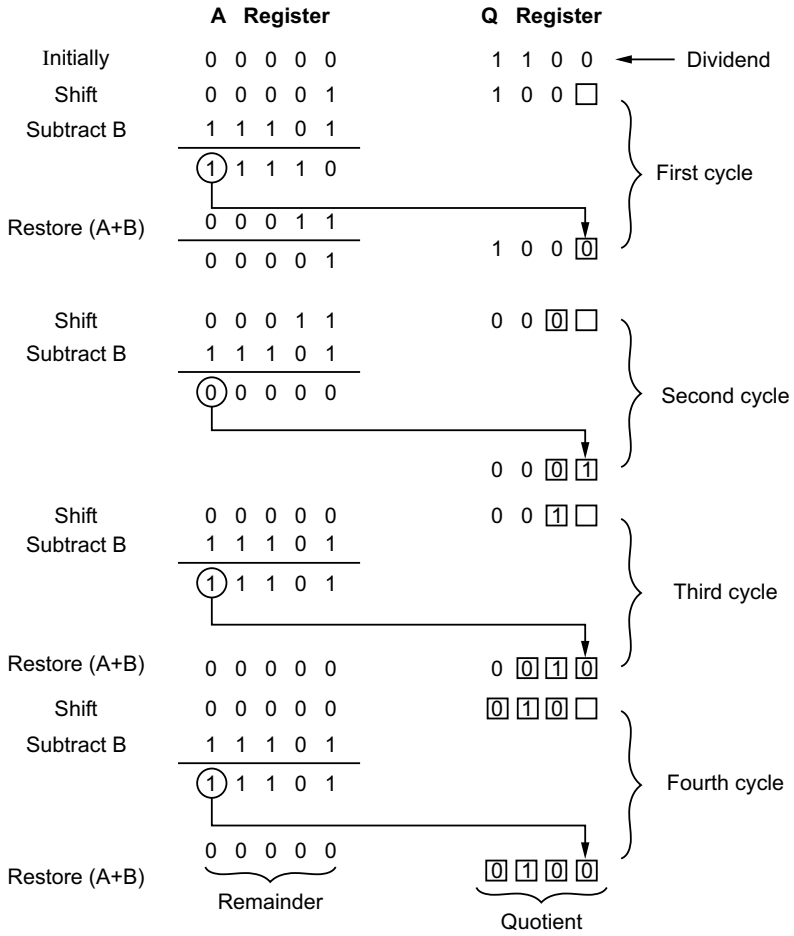


Fig. 4.13.5

Example for Practice

Example 4.13.2 *Demonstrate the division of 1100_2 by 101_2 using restoring method, draw block diagram and explain the operation.* **Dec.-08, Marks 16**

4.13.2 Non-Restoring Division

A flowchart for non-restoring division operation is as shown in Fig. 4.13.6. Let us see one example. Consider 4-bit dividend and 2-bit divisor :

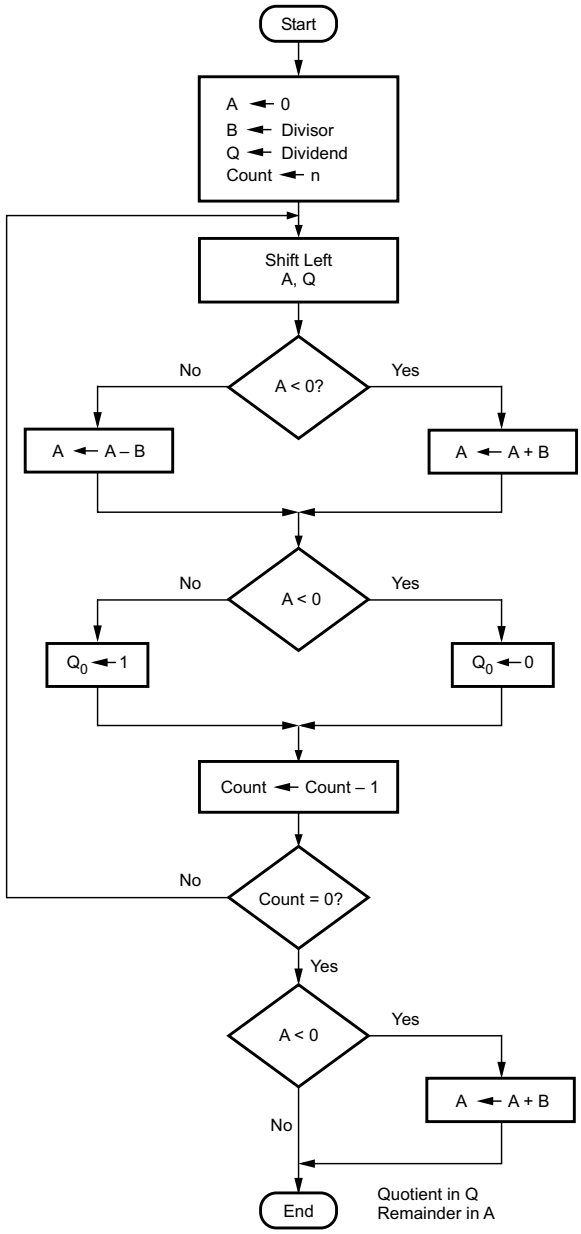


Fig. 4.13.6 Flowchart for non-restoring division operation

Dividend = 1 0 1 0

Divisor = 0 0 1 1

Fig. 4.13.7 shows steps involved in the non-restoring binary division.

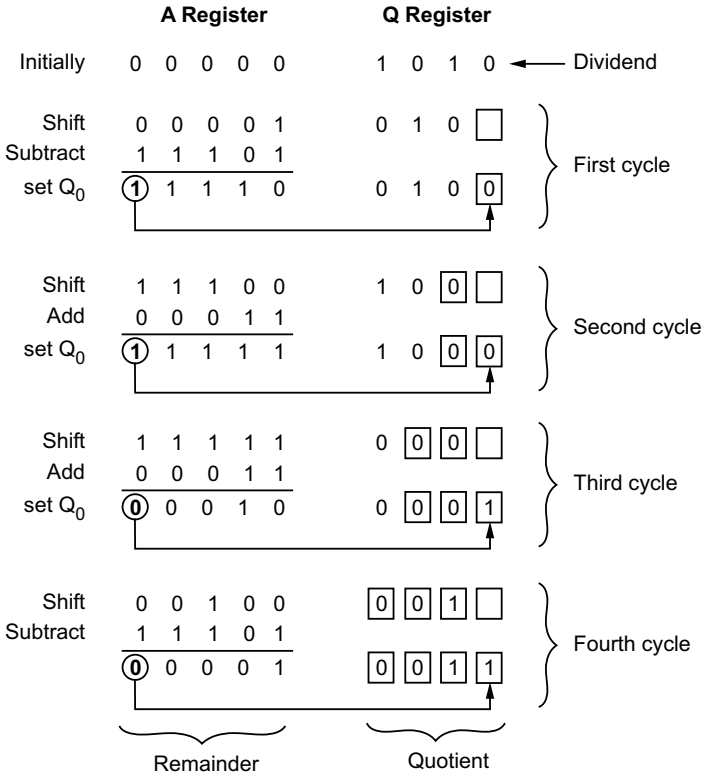


Fig. 4.13.7 A non-restoring division example

In the above example after 4 cycles register A is positive and hence step 3 is not required. Let us see another example where we need step 3. Consider 4-bit dividend and 2-bit divisor :

Dividend = 1 0 1 1

Divisor = 0 1 0 1

Fig. 4.13.8 shows steps involved in the non-restoring binary division.

The hardware shown in Fig. 4.13.2 can also be used to perform non-restoring algorithm. There is no simple algorithm for signed division. In signed division, the operands are preprocessed to transform them into positive values. Then using one of the algorithms just discussed quotients and remainders are calculated. The quotients and remainders are then transformed to the correct signed values.

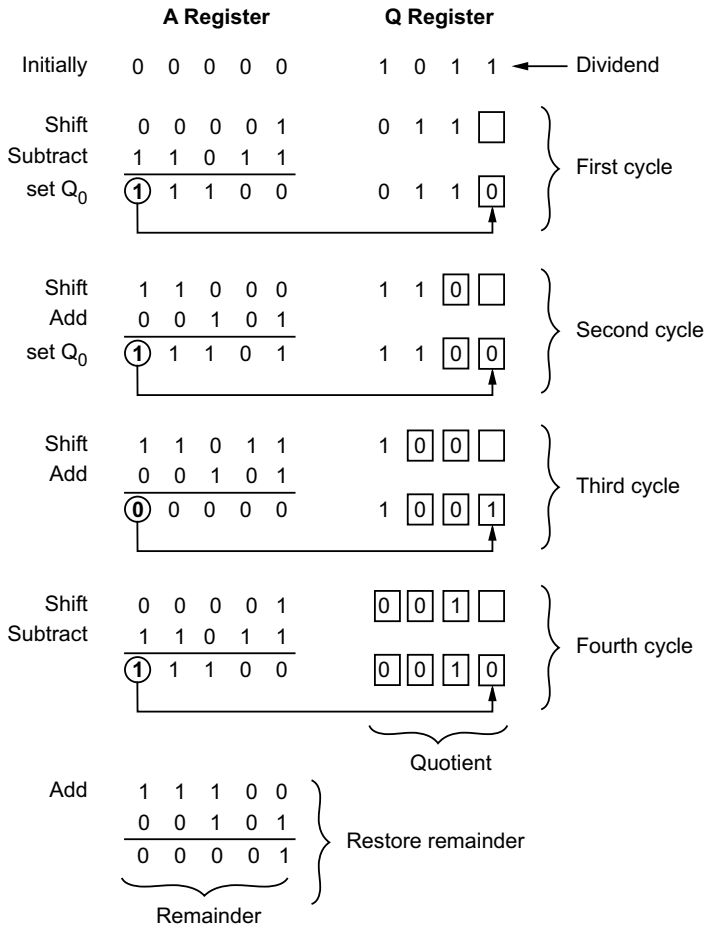


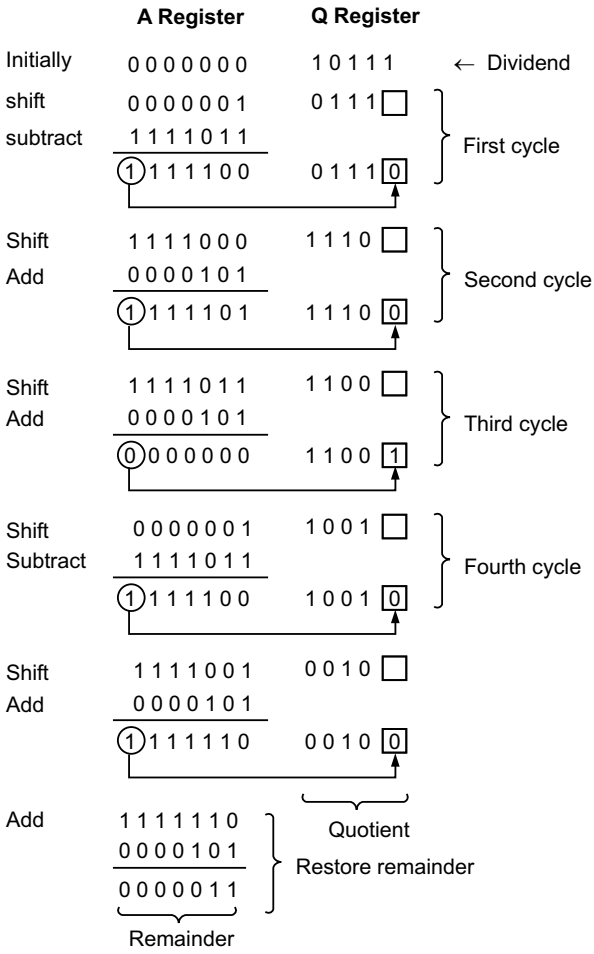
Fig. 4.13.8 A non-restoring division example

Example 4.13.3 *Using the non-restoring division algorithm, compute 23/5.* **May-07, Marks 8**

Solution :

$$\text{Dividend} = 23_{10} = 10111$$

$$\text{Divisor} = 5_{10} = 00101$$



Example 4.13.4 *Perform 1100 ÷ 11 using non-restoring algorithm.*

Solution : Non-Restoring Division : Q ← Dividend (1100), B ← Divisor (11).

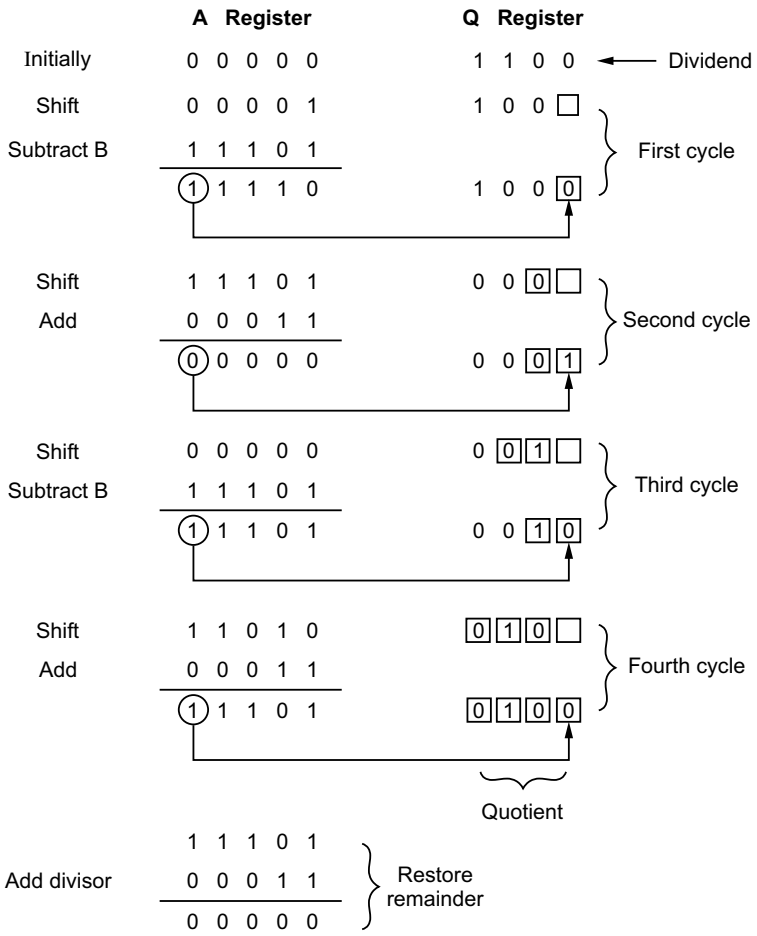


Fig. 4.13.9

Examples for Practice

Example 4.13.5 *Explain non-restoring division algorithm with the help of suitable example.* **May-09, Marks 8**

Example 4.13.6 *Perform the division on the following 5-bit unsigned integer using non-restoring division : 10101/00101.* **May-06, Marks 4**

4.13.3 Comparison between Restoring and Non-Restoring Division Algorithm

Sr. No.	Restoring	Non-restoring
1.	Needs restoring of register A if the result of subtraction is negative.	Does not need restoring.
2.	In each cycle content of register A is first shifted left and then divisor is subtracted from it.	In each cycle content of register A is first shifted left and then divisor is added or subtracted with the content of register A depending on the sign of A.
3.	Does not need restoring of remainder.	Needs restoring of remainder if remainder is negative.
4.	Slower algorithm.	Faster algorithm.

Review Questions

1. *Describe the algorithm for integer division with suitable example.* **May-07, Dec.-07, 09, Marks 8**
2. *Explain the restoring division algorithm.* **May-09, Marks 8**
3. *Explain non-restoring division algorithm with the help of suitable example.* **May-09, Marks 8**
4. *Give the comparison between restoring and non-restoring division.*

4.14 Multiplexer

May-07, 08, 09, 11, Dec.-03, 06, 08, 09, 11

In digital systems, many times it is necessary to select single data line from several data-input lines, and the data from the selected data line should be available on the output. The digital circuit which does this task is a multiplexer.

It is a digital switch. It allows digital information from several sources to be routed onto a single output line, as shown in the Fig. 4.14.1. The basic multiplexer has several data-input lines and a single output line. The selection of a particular input line is controlled by a set of selection lines. Since multiplexer selects one of the input and routes it to output, it is also known as **data selector**. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. Therefore, multiplexer is '**many into one**' and it provides the digital equivalent of an analog selector switch.

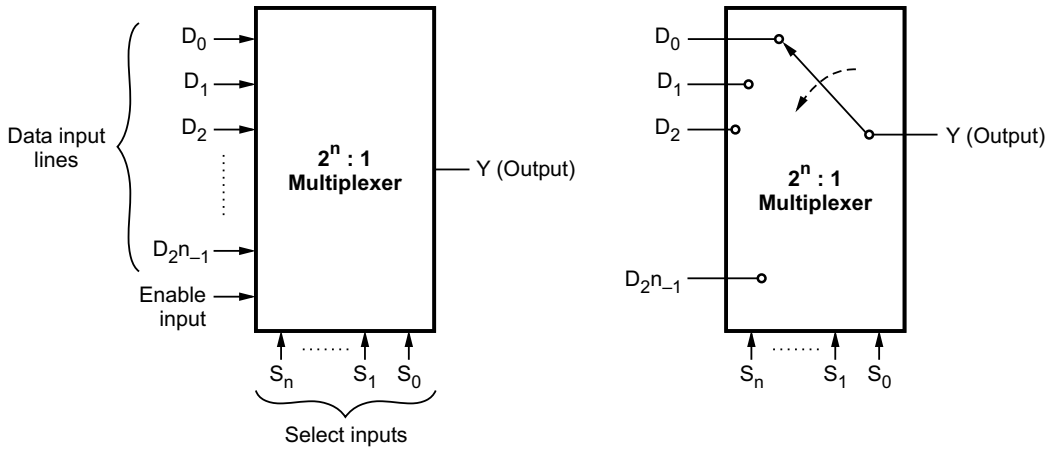
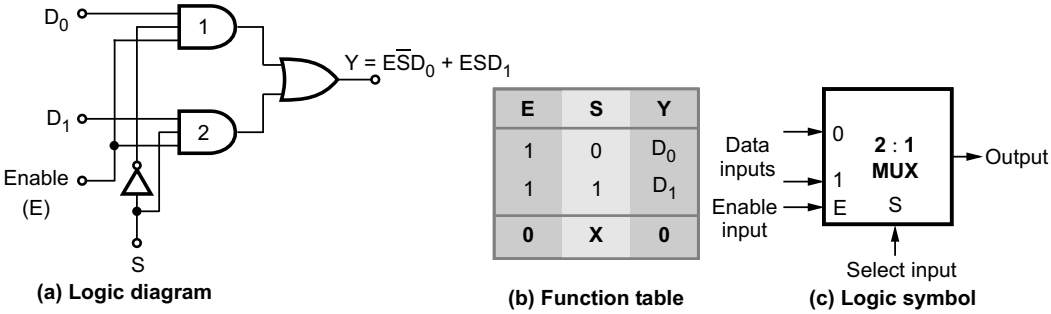


Fig. 4.14.1

4.14.1 2 : 1 Multiplexer

Fig. 4.14.2 (a) shows 2 : 1 multiplexer. D_0 is applied as an input to one AND gate and D_1 is applied as an input to another AND gate. Enable input is applied to both gates as one input. Selection line S is connected as second input to second AND gate. An inverted S is applied as second input to first AND gate. Outputs of both AND gates are applied as inputs to OR gate.



E	S	Y
1	0	D_0
1	1	D_1
0	X	0

(b) Function table

Fig. 4.14.2

Working

When E = 0, output is 0, i.e. Y = 0 irrespective of any input condition. When E = 1 the circuit works as follows :

When S = 0, the inverted S, that is 1 gets applied as second input to first AND gate. Since S is applied directly as input to second AND gate; its output goes zero irrespective

of first input. Since the second input of first AND gate is 1, its output is equal to its first input, that is D_0. Hence $Y = D_0$.

Exactly opposite is the case when S = 1. In this case, second AND gate output is equal to its first input D_1 and first AND gate output is 0. Hence $Y = D_1$. Both these cases are summarized in truth table shown in Fig. 4.14.2 (b).

Deriving realization expression

The Table 4.14.1 shows the truth table for 2 : 1 multiplexer. From the truth table it is clear that Y = 1 when $E\,\overline{S}\,D_0 = 1$ or $E\,S\,D_1 = 1$ as indicated by shaded rows.

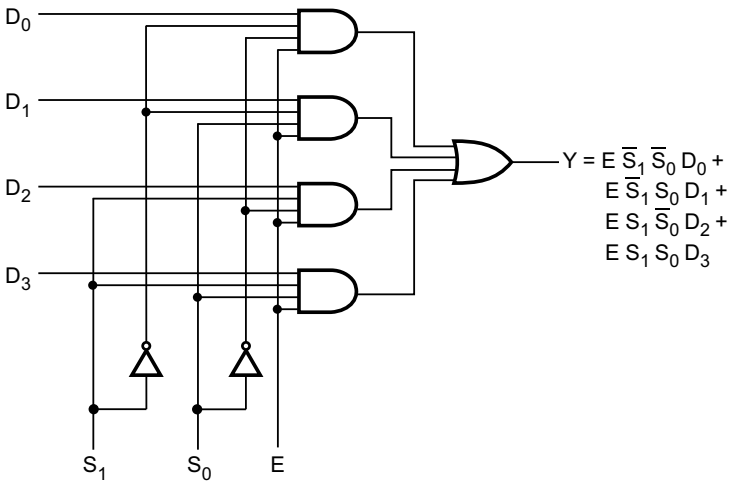
$$\therefore \quad Y = E\,\overline{S}\,D_0 + E\,S\,D_1$$

Enable (E)	Select (S)	D_1	D_0	Output Y	
1	0	X	0	0	
1	0	X	1	1	$E\overline{S}\,D_0$
1	1	0	X	0	
1	1	1	X	1	$ES\,D_1$
0	X	X	X	0	

Table 4.14.1 Truth table for 2 : 1 multiplexer

4.14.2 4 : 1 Multiplexer

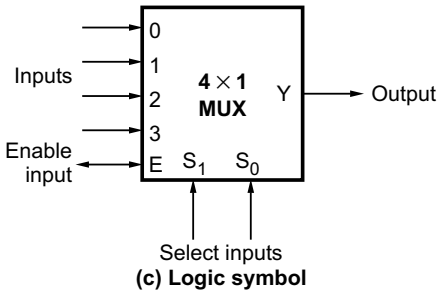
Fig. 4.14.3 (a) shows 4-to-1 line multiplexer. Each of the four lines, D_0 to D_3, is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



(a) Logic diagram

E	S_1	S_0	Y
1	0	0	D_0
1	0	1	D_1
1	1	0	D_2
1	1	1	D_3
0	X	X	0

(b) Function table



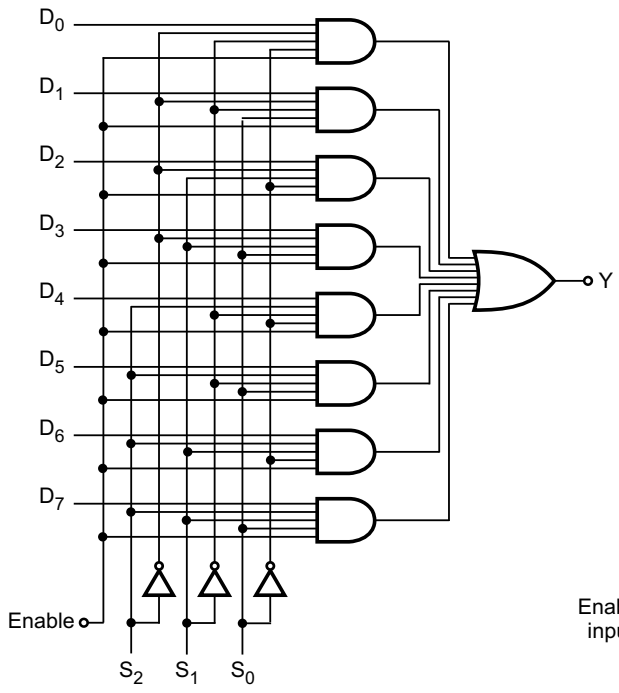
(c) Logic symbol

Fig. 4.14.3 4 to 1 line multiplexer

For example, when S_1S_0 = 0 1, the AND gate associated with data input D_1 has two of its inputs equal to 1 and the third input connected to D_1. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of D_1, thus we can say data bit D_1 is routed to the output when S_1 S_0 = 0 1.

4.14.3 8 : 1 Multiplexer

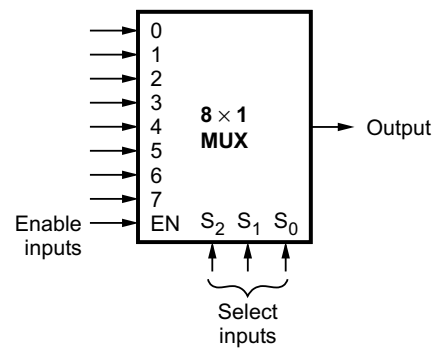
Fig. 4.14.4 shows 8 : 1 multiplexer.



(a) Logic diagram

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

(b) Function table



(c) Logic symbol

Fig. 4.14.4 8 : 1 Multiplexer

There are eight input lines one output line and three select lines. As shown in the function table, the selection of a particular input line is controlled by three selection lines.

4.14.4 Quadruple 2 to 1 Multiplexer

In some cases, two or more multiplexers are enclosed within one IC package, as shown in the Fig. 4.14.5. The Fig. 4.14.5 shows quadruple 2-to-1 line multiplexer, i.e. four multiplexers, each capable of selecting one of two input lines. Output Y_1 can be selected to be equal to either A_1 or B_1. Similarly output Y_2 may have the value of A_2 or B_2, and so on. The selection line S selects one of two lines in all four multiplexers. The control input E enables the multiplexers in the 0 state and disables them in the 1 state. When E = 1, outputs have all 0's, regardless of the value of S.

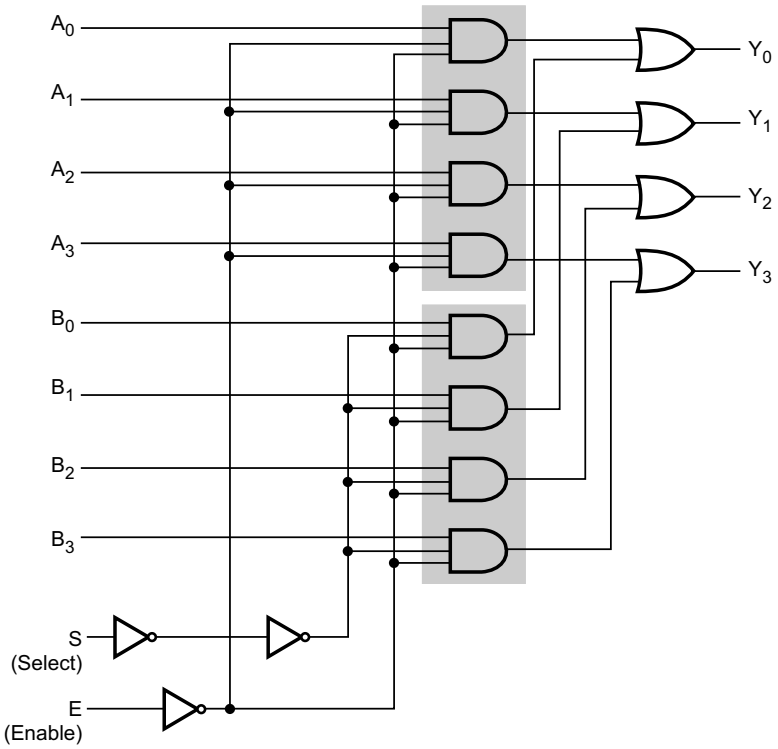


Fig. 4.14.5 Quadruple 2-to-1 line multiplexer

Function Table

E	S	Output Y
1	X	All 0s
0	0	Select A
0	1	Select B

4.14.5 Expanding Multiplexers

It is possible to expand range of inputs for multiplexer beyond the available range by interconnecting several multiplexers in cascade. The circuit with two or more multiplexers connected to obtain the multiplexer with more number of inputs is known as **multiplexer tree.**

Examples for Understanding

Example 4.14.1 *Design 16 : 1 multiplexer using 8 : 1 multiplexer.*

Solution :

Step 1 : Connect the select lines (S_2, S_1 and S_0) of two multiplexers in parallel.

Step 2 : Connect most significant select line (S_3) such that when $S_3 = 0$ MUX1 is enabled and when $S_3 = 1$, MUX2 is enabled.

Step 3 : Logically OR the outputs of two multiplexers to obtain the final output Y.

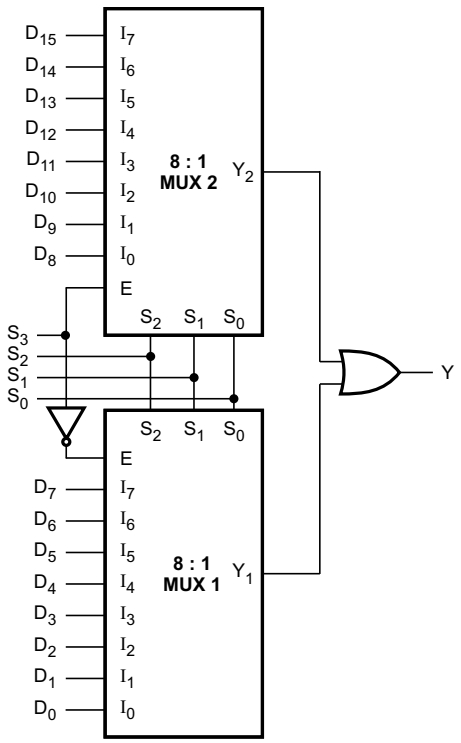


Fig. 4.14.6 16 : 1 multiplexer using two 8 : 1 multiplexers

Example 4.14.2 *Design 16 : 1 multiplexer using 4 : 1 multiplexers.*

Solution : Since there are 16-inputs for the multiplexers we require four 4 : 1 multiplexers to satisfy input needs. The four outputs of 4 : 1 multiplexers are again multiplexed by 4 : 1 multiplexer to generate final output.

Step 1 : Connect the select lines (S_1 and S_0) of four multiplexers in parallel.

Step 2 : Connect the most significant select lines (S_3 and S_2) to the MUX5.

Step 3 : Connect the outputs Y_0, Y_1, Y_2 and Y_4 of four multiplexers as data inputs for the MUX 5, as shown in the Fig. 4.14.7.

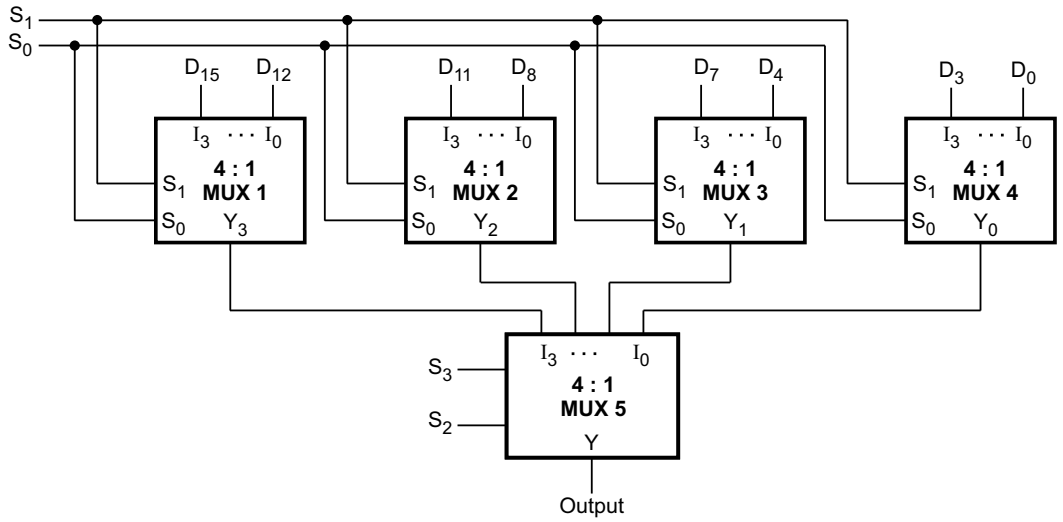


Fig. 4.14.7

Examples with Solutions

Example 4.14.3 *Draw 32 : 1 multiplexer using two 16 : 1 multiplexers and one 2 : 1 multiplexer.*

Solution : The Fig. 4.14.8 shows the connection diagram for 32 : 1 multiplexer using two 16 : 1 multiplexer and one 2 : 1 multiplexer. Here, select lines S_0, S_1, S_2 and S_3 are

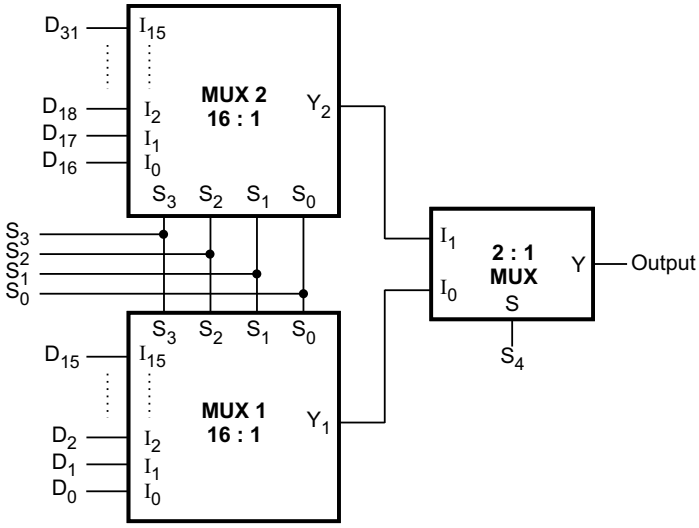


Fig. 4.14.8

connected in parallel to 16 : 1 multiplexers and select line S_4 is used to select either Y_1 or Y_2 using 2 : 1 multiplexer.

Example 4.14.4 *Draw 64 : 1 multiplexer tree using 16 : 1 multiplexer.*

Solution : The Fig. 4.14.9 shows 64 : 1 multiplexer using four 16 : 1 multiplexers and one 4 : 1 multiplexer.

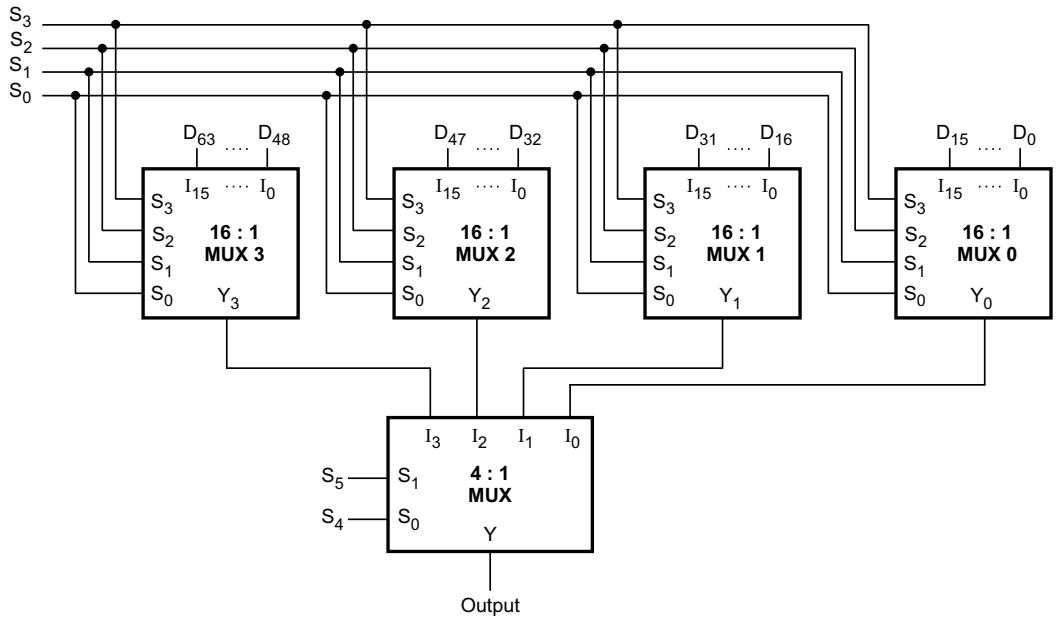


Fig. 4.14.9

Examples for Practice

Example 4.14.5 *Draw the block diagram of a 4 : 1 multiplexer using 2 : 1 MUX.*

Example 4.14.6 *Design 32 : 1 MUX using 8 : 1 MUX.*

4.14.6 Implementation of Combinational Logic using MUX

A multiplexer consists of a set of AND gates whose outputs are connected to single OR gate. Because of this construction any Boolean function in a SOP form can be easily realized using multiplexer. Each AND gate in the multiplexer represents a minterm. In 8 to 1 multiplexer, there are 3 select inputs and 23 minterms. By connecting the function variables directly to the select inputs, a multiplexer can be made to select the AND gate that corresponds to the minterm in the function. If a minterm exists in a function, we have to connect the AND gate data input to logic 1; otherwise we have to connect it to logic 0. This is illustrated in the following example.

Examples for Understanding

Example 4.14.7 *Implement the given function using multiplexer.*
$F(x, y, z) = \Sigma(0, 2, 6, 7)$. **May-07, Marks 10**

Solution :

Step 1 : Select the multiplexer. Here, Boolean expression has 3 variables, thus we require $2^3 = 8 : 1$ multiplexer.

Step 2 : Connect inputs corresponds to the present minterms to logic 1.

Step 3 : Connect remaining inputs to logic 0.

Step 4 : Connect input variables to select lines of MUX.

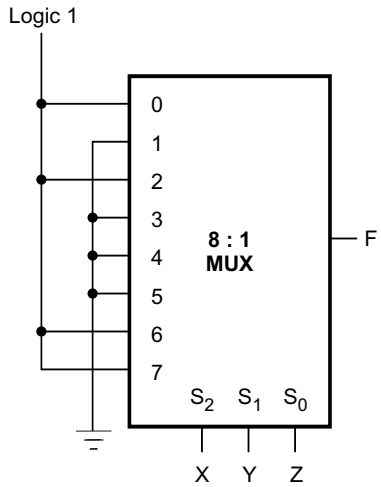


Fig. 4.14.10

Example 4.14.8 *Implement the Boolean function represented by the given truth table using multiplexer.*

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Solution :

Step 1 : Select the multiplexer. Here, there are three input variables, thus we require $2^3 = 8 : 1$ multiplexer.

Step 2 : Find the minterm expression.

Minterm expression for given truth table is $\sum$ m (1, 2, 5, 7).

Step 3 : Connect inputs corresponds to the present minterms to logic 1.

Step 4 : Connect remaining inputs to logic 0.

Step 5 : Connect input variables to select lines of MUX.

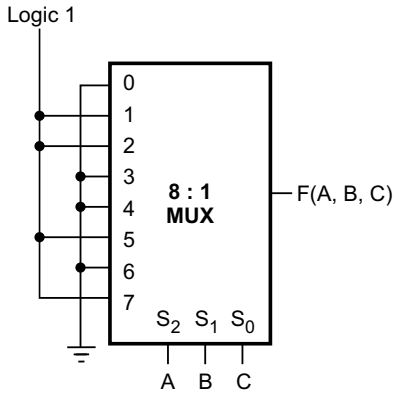


Fig. 4.14.11

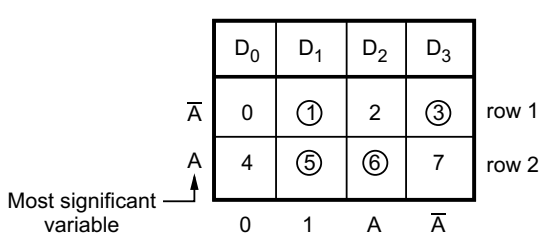
In the above example, we have seen the method for implementing Boolean function of 3 variables with $2^3(8)$ - to - 1 multiplexer. Similarly, we can implement any Boolean function of n variables with 2^n-to-1 multiplexer. However, it is possible to do better than this. If we have Boolean function of n + 1 variables, we take n of these variables and connect them to the selection lines of a multiplexer. The remaining single variable of the function is used for the inputs of the multiplexer. In this way we can implement any Boolean function of n variables with 2^{n-1}- to - 1 multiplexer. Let us see some example.

Example 4.14.9 *Implement the following Boolean function using 4 : 1 multiplexer.*
F (A, B, C) = $\sum$ m (1, 3, 5, 6)

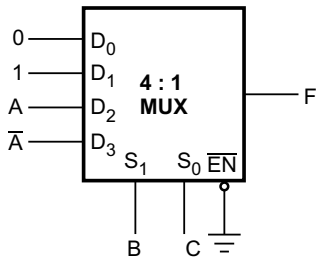
Solution :

Step 1 : Connect least significant variables as a select inputs of multiplexer. Here, connect C to S_0 and B to S_1.

Step 2 : Derive inputs for multiplexer using implementation table.



(a) Implementation table



(b) Multiplexer implementation

Fig. 4.14.12

As shown in the Fig. 4.14.12 (a) the implementation table is nothing but the list of the inputs of the multiplexer and under them list of all the minterms in two rows. The first row lists all those minterms where A is complemented, and the second row lists all the minterms with A uncomplemented. The minterms given in the function are circled and then each column is inspected separately as follows :

- If the two minterms in a column are not circled, 0 is applied to the corresponding multiplexer input (see column 0).
- If the two minterms in a column are circled, 1 is applied to the corresponding multiplexer input (see column 1).
- If the minterm in the second row is circled and minterm in the first row is not circled, A is applied to the corresponding multiplexer input (see column 2).
- If the minterm in the first row is circled and minterm in the second row is not circled, $\overline{A}$ is applied to the corresponding multiplexer input (see column 3).

Example 4.14.10 *Implement the following Boolean function using 8 : 1 multiplexer*
$F(A, B, C, D) = \overline{A}\ B\ \overline{D} + A\,C\,D + \overline{B}\ C\ D + \overline{A}\ \overline{C}\ D$

Solution :

Step 1 : Express Boolean function in the minterm form.

The given Boolean expression is not in standard SOP form. Let us first convert this in standard SOP form

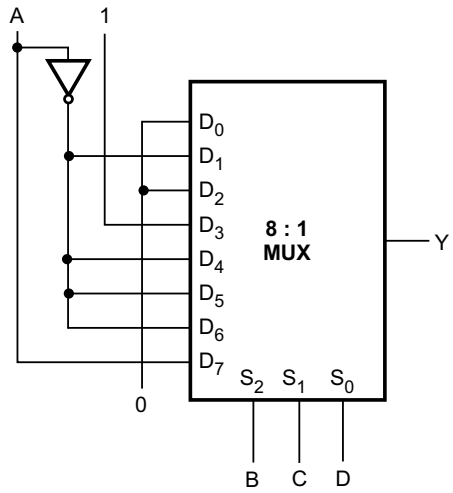
$$\begin{aligned} F(A, B, C, D) &= \overline{A}\,B\,\overline{D}\,(C+\overline{C}) + A\,C\,D(B+\overline{B}) + \overline{B}\,C\,D\,(A+\overline{A}) + \overline{A}\,\overline{C}\,D(B+\overline{B}) \\ &= \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,C\,D + A\,\overline{B}\,C\,D \\ &\quad + A\,\overline{B}\,C\,D + \overline{A}\,\overline{B}C\,D + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,\overline{B}\,\overline{C}\,D \\ &= \overline{A}\,B\,C\,\overline{D} + \overline{A}\,B\,\overline{C}\,\overline{D} + A\,B\,C\,D + A\,\overline{B}\,C\,D + \overline{A}\,\overline{B}C\,D + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,\overline{B}\,\overline{C}\,D \\ &= \sum m\,(6, 4, 15, 11, 3, 5, 1) \\ &= \sum m\,(1, 3, 4, 5, 6, 11, 15) \end{aligned}$$

Step 2 : Implement it using implementation table.

From the Boolean function in the minterm form can be implemented using 8 : 1 multiplexer as follows :

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	0	①	2	③	④	⑤	⑥	7
A	8	9	10	⑪	12	13	14	⑮
	0	$\overline{A}$	0	1	$\overline{A}$	$\overline{A}$	$\overline{A}$	A

(a) Implementation table



(b) Multiplexer implementation

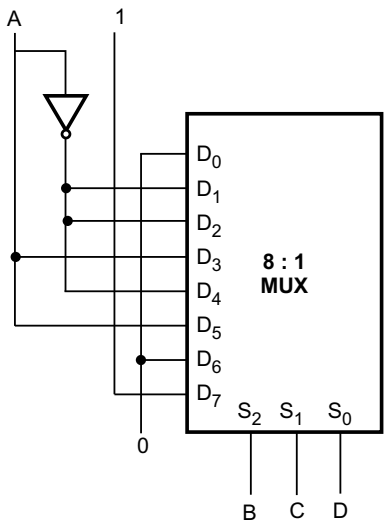
Fig. 4.14.13

Example 4.14.11 *Implement the following Boolean function with 8 : 1 multiplexer*
$F(A, B, C, D) = \pi M (0, 3, 5, 8, 9, 10, 12, 14)$

Solution : Here, instead of minterms, maxterms are specified. Thus, we have to circle maxterms which are not included in the Boolean function. Fig. 4.14.14 shows the implementation of Boolean function with 8 : 1 multiplexer.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	0	①	②	3	④	5	6	⑦
A	8	9	10	⑪	12	⑬	14	⑮
	0	$\overline{A}$	$\overline{A}$	A	$\overline{A}$	A	0	1

(a) Implementation table



(b) Multiplexer implementation

Fig. 4.14.14

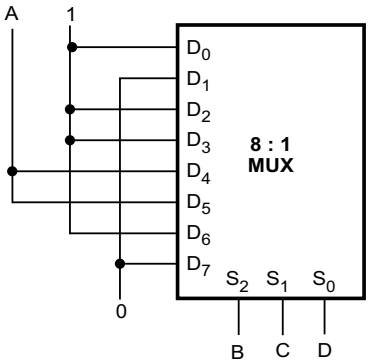
Example 4.14.12 *Implement the following Boolean function with 8 : 1 multiplexer.*
$F(A, B, C, D) = \sum m(0, 2, 6, 10, 11, 12, 13) + d(3, 8, 14)$

Solution : In the given Boolean function three don't care conditions are also specified. We know that don't care conditions can be treated as either 0s or 1s. Fig. 4.14.15 shows the implementation of given Boolean function using 8 : 1 multiplexer.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	(0)	1	(2)	(3)	4	5	(6)	7
A	(8)	9	(10)	(11)	(12)	(13)	(14)	15
	1	0	1	1	A	A	1	0

Here, don't cares are treated as 1s

(a) Implementation table



(b) Implementation

Fig. 4.14.15

In this example, by taking don't care conditions 8 and 14 as 1s we have eliminated $\overline{A}$ term and hence the inverter.

Examples with Solutions

Example 4.14.13 *Implement full adder circuit using 8 : 1 multiplexer.* **May-11, Marks 5**

Solution :

Inputs			Outputs	
A	**B**	**C_{in}**	**Carry**	**Sum**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.14.2

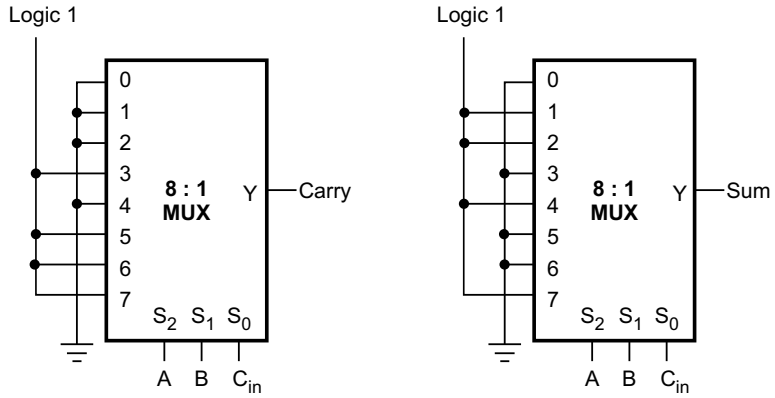


Fig. 4.14.16

Example 4.14.14 *Implement full adder circuit using quadruple 2 to 1 multiplexer.*

Solution : Implementation tables :

Carry

	D_0	D_1	D_2	D_3
$\overline{A}$	0	0	0	1
A	0	1	1	1
	0	A	A	1

Sum

	D_0	D_1	D_2	D_3
$\overline{A}$	0	1	1	0
A	1	0	0	1
	A	$\overline{A}$	$\overline{A}$	A

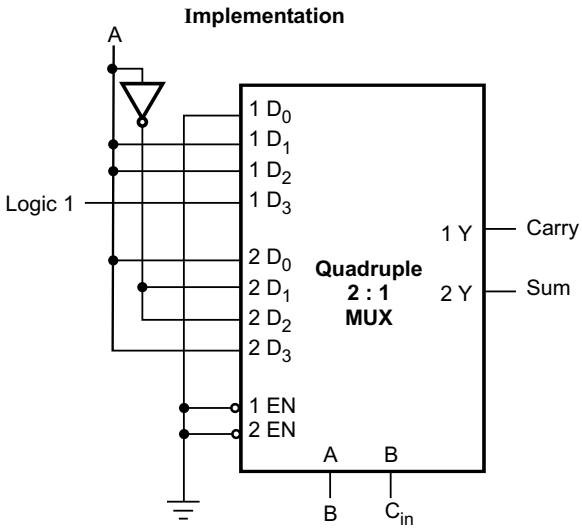


Fig. 4.14.17

Example 4.14.15 *Realize* $F(w, x, y, z) = \sum (1, 4, 6, 7, 8, 9, 10, 11, 15)$ *using 4 to 1 MUX.*

May-04, Marks 8

Solution :

	D_0	D_1	D_2	D_3
$\overline{w}\,\overline{x}$	0	(1)	2	3
$\overline{w}\,x$	(4)	5	(6)	(7)
$w\,\overline{x}$	(8)	(9)	(10)	(11)
$w\,x$	12	13	14	(15)

$$D_0 = \overline{w}\,x + w\,\overline{x} = w \oplus x \qquad D_1 = \overline{w}\,\overline{x} + \overline{w}\,\overline{x} = \overline{x}$$

$$D_2 = \overline{w}\,x + w\,\overline{x} = w \oplus x \qquad D_3 = \overline{w}\,x + w\,\overline{x} + w\,x = x + w\,\overline{x} = w + x$$

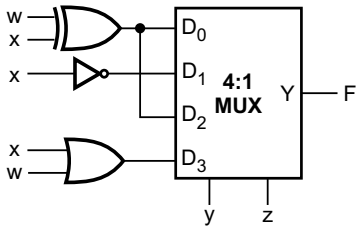


Fig. 4.14.18

Example 4.14.16 *Implement the function with a multiplexer.*

$F(A, B, C, D) = \sum (0,1,3,4,8,9,15)$ **Dec.-08, Marks 6**

Solution : Implementation table :

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{A}$	(0)	(1)	2	(3)	(4)	5	6	7
A	(8)	(9)	10	11	12	13	14	(15)
	1	1	0	$\overline{A}$	$\overline{A}$	0	0	A

Implementation

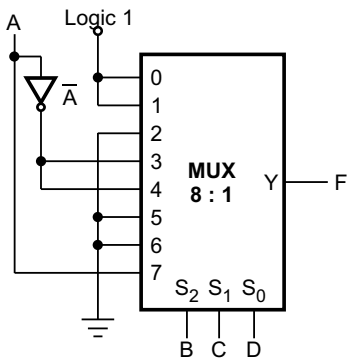


Fig. 4.14.19

Example 4.14.17 *Implement the following function using suitable multiplexer.*

$F(x, y, z) = \sum m\ (0, 2, 5, 7)$ **May-09, Marks 2**

Solution :

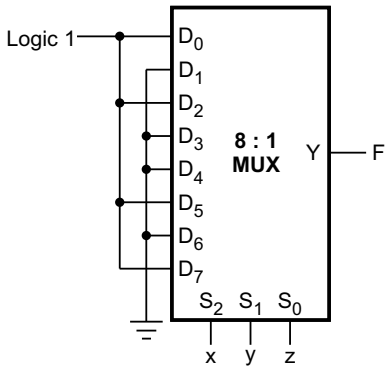


Fig. 4.14.20

Example 4.14.18 *Implement the given Boolean function using 8 : 1 multiplexer.*

$F(A, B, C) = \sum(1, 3, 5, 6)$ **Dec.-09, Marks 6**

Solution :

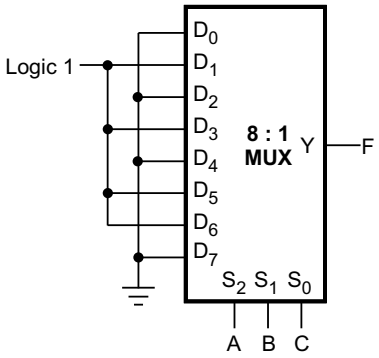


Fig. 4.14.21

Example 4.14.19 *Implement the following function using suitable multiplexer*

$F(A, B, C, D) = \sum\ (1,\ 3,\ 4,\ 11,\ 12,\ 13,\ 14,\ 15)$ **May-11, Marks 8**

Solution :

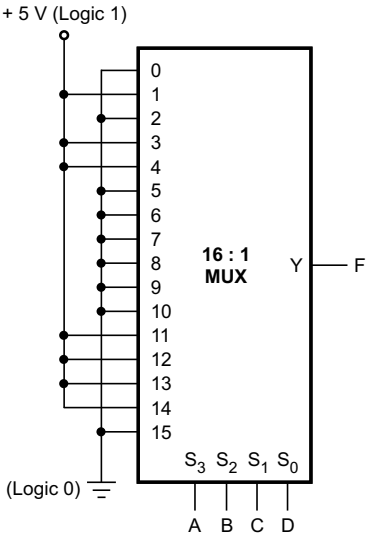


Fig. 4.14.22

Example 4.14.20 *Realize the Boolean function using appropriate multiplexer* $F(A, B, C) = \sum (0, 1, 3, 7)$ **Dec.-11, Marks 2**

Solution :

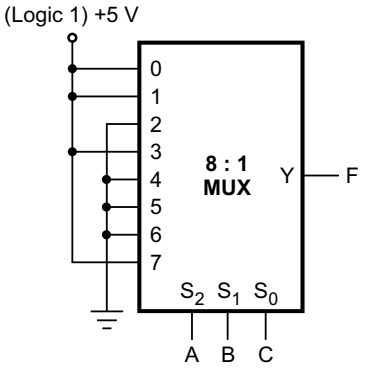


Fig. 4.14.23

Examples for Practice

Example 4.14.21 *Implement the following function using 8:1 multiplexer* $f(a, b, c, d) = \sum m\ (0, 1, 5, 6, 8, 10, 12, 15)$

Example 4.14.22 *Implement Boolean function* $f = AB + \overline{C}D + A\overline{B}C$ *using 8 : 1 multiplexer.*

4.14.7 Applications of Multiplexer

1. They are used as a data selector to select one out of many data inputs.
2. They can be used to implement combinational logic circuit.
3. They are used in time multiplexing systems.
4. They are used in frequency multiplexing systems.
5. They are used in A/D and D/A converter.
6. They are used in data acquisition systems.

4.14.8 Multiplexer ICs

IC number	Function
74150	16 : 1 multiplexer
74151	8 : 1 multiplexer
74153	Dual 4 : 1 multiplexer
74157	Quad 2-input multiplexer
Refer Appendix A for details	

Table 4.14.3 Multiplexer ICs

Review Questions

1. *Define multiplexer.*
2. *What is data selector.* **Dec.-06, Marks 2**
3. *Draw and explain the working of 2 : 1 multiplexer and realize it using basic gates.* **Dec.-03, Marks 2**
4. *Explain 4 : 1 multiplexer with the help of logic circuit and truth table.* **Dec.-03, Marks 8**
5. *Write the truth table of a 4 : 1 multiplexer.* **May-08, Marks 2**
6. *Explain 8 : 1 multiplexer with the help of logic circuit and truth table.*
7. *Explain the concept and working of quadruple 2 to 1 line multiplexer.* **May-07, Marks 6**
8. *State the applications of multiplexers.*

4.15 Demultiplexer **May-04**

A demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of specific output line is controlled by the values of n selection lines.

The Fig. 4.15.1 shows the block diagram of a demultiplexer. It has one input data line, 2^n output lines, n select lines and one enable input.

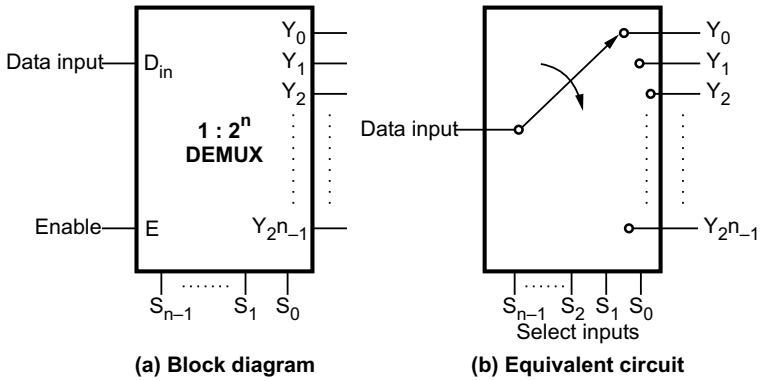


Fig. 4.15.1

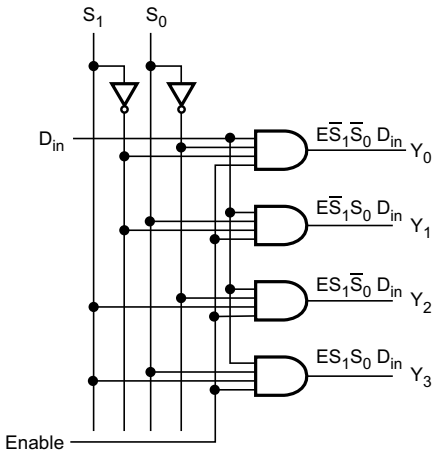
Differentiate between Multiplexer and Demultiplexer

Parameter	Multiplexer	Demultiplexer
Definition	Multiplexer is a digital switch which allows digital information from several sources to be routed onto a single output line.	Demultiplexer is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines
Number of data inputs	2^n	1
Number of data outputs	1	2^n
Relationship of input and output	Many to one	One to many
Applications	• Used as a data selector • In time division multiplexing at the transmitting end	• Used as a data distributor • In time division multiplexing at the receiving end

4.15.1 Types of Demultiplexers

4.15.1.1 1 : 4 Demultiplexer

Fig. 4.15.2 shows 1 : 4 demultiplexer. The single input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines depending on the select inputs. Enable input should be high to enable demultiplexer.



(a) Logic diagram

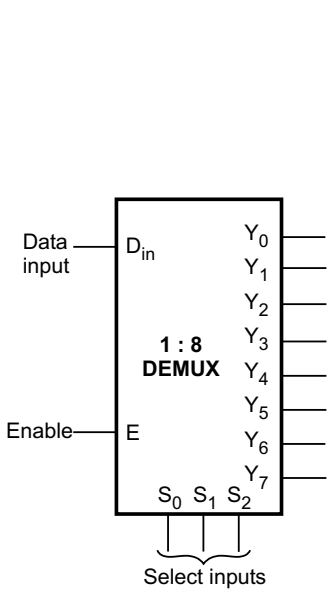
Enable (E)	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3	
0	X	X	X	0	0	0	0	
1	0	0	0	0	0	0	0	D_{in} is routed to Y_0
1	0	0	1	1	0	0	0	$Y_0 = E\,\overline{S}_1\,\overline{S}_0\,D_{in}$
1	0	1	0	0	0	0	0	D_{in} is routed to Y_1
1	0	1	1	0	1	0	0	$Y_1 = E\,\overline{S}_1\,S_0\,D_{in}$
1	1	0	0	0	0	0	0	D_{in} is routed to Y_2
1	1	0	1	0	0	1	0	$Y_2 = E\,S_1\,\overline{S}_0\,D_{in}$
1	1	1	0	0	0	0	0	D_{in} is routed to Y_3
1	1	1	1	0	0	0	1	$Y_3 = E\,S_1\,S_0\,D_{in}$

(b) Function table for 1 : 4 demultiplexer

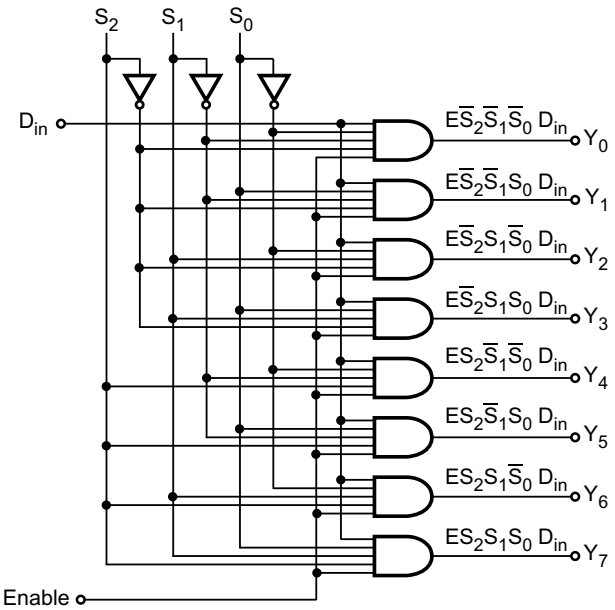
Fig. 4.15.2

4.15.1.2 1 : 8 Demultiplexer

The Fig. 4.15.3 shows 1 : 8 demultiplexer. The single input data D_{in} has a path to all eight outputs, but the input information is directed to only one of the output lines depending on the select inputs.



(a) Block schematic



(b) Logic diagram

Enable	Select inputs			Outputs								
E	S_2	S_1	S_0	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7	
0	X	X	X	0	0	0	0	0	0	0	0	
1	0	0	0	D_{in}	0	0	0	0	0	0	0	$E\ \bar{S}_2\ \bar{S}_1\ \bar{S}_0\ D_{in}$
1	0	0	1	0	D_{in}	0	0	0	0	0	0	$E\ \bar{S}_2\ \bar{S}_1\ S_0\ D_{in}$
1	0	1	0	0	0	D_{in}	0	0	0	0	0	$E\ \bar{S}_2\ S_1\ \bar{S}_0\ D_{in}$
1	0	1	1	0	0	0	D_{in}	0	0	0	0	$E\ \bar{S}_2\ S_1\ S_0\ D_{in}$
1	1	0	0	0	0	0	0	D_{in}	0	0	0	$E\ S_2\ \bar{S}_1\ \bar{S}_0\ D_{in}$
1	1	0	1	0	0	0	0	0	D_{in}	0	0	$E\ S_2\ \bar{S}_1\ S_0\ D_{in}$
1	1	1	0	0	0	0	0	0	0	D_{in}	0	$E\ S_2\ S_1\ \bar{S}_0\ D_{in}$
1	1	1	1	0	0	0	0	0	0	0	D_{in}	$E\ S_2\ S_1\ S_0\ D_{in}$

(c) Function table

Fig. 4.15.3 1 : 8 demultiplexer

Example 4.15.1 *Draw the diagram and explain 1 to 16 demultiplexer circuit.*

May-07, Marks 8

Solution : (Fig. 4.15.4 see on next page)

4.15.2 Expanding Demultiplexers

To provide larger output needs we can cascade two or more demultiplexer to get demultiplexer with more number of output lines. Such a connection is known as **demultiplexer tree.**

Example 4.15.2 *Design 1 : 8 demultiplexer using two 1 : 4 demultiplexers.*

Solution :

Step 1 : Connect D_{in} signal to D_{in} input of both the demultiplexers.

Step 2 : Connect select lines B and C to select lines S_1 and S_0 of the both demultiplexers, respectively.

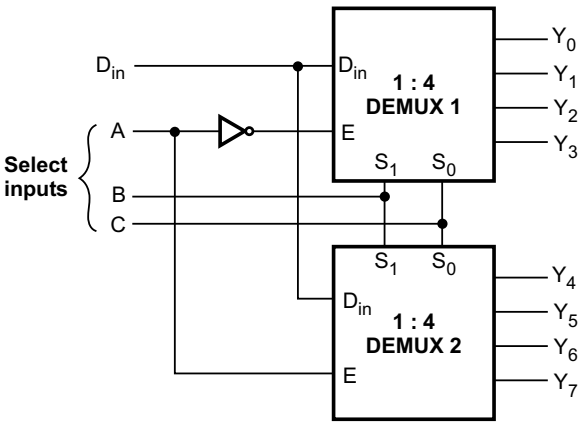


Fig. 4.15.5 Cascading of demultiplexers

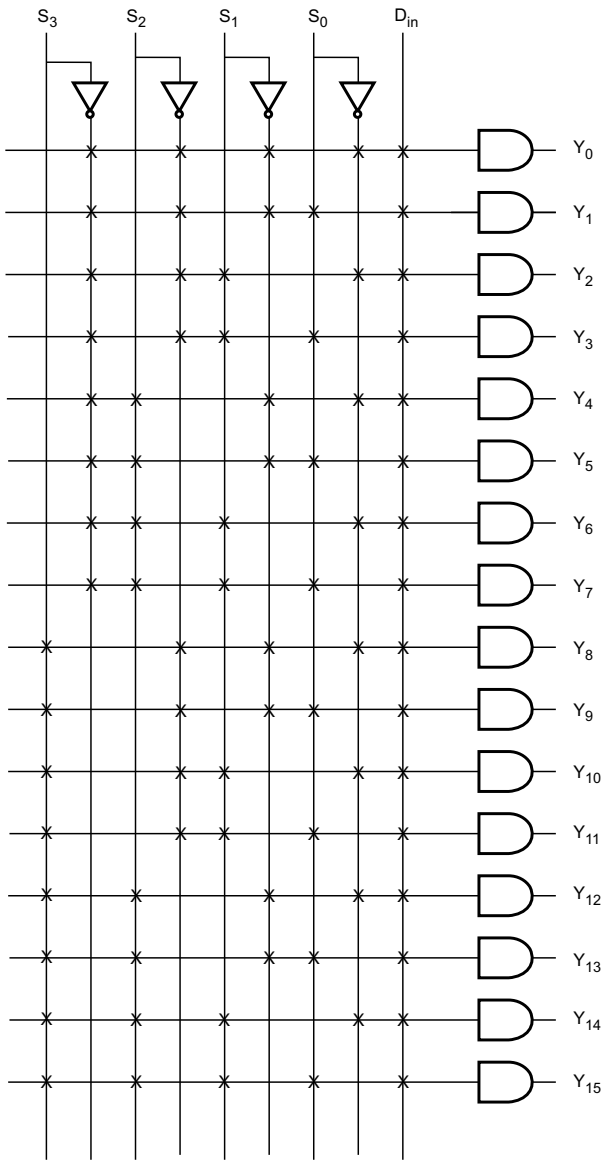


Fig. 4.15.4 1 to 16 Demultiplexer

Step 3 : Connect most significant select line (A) such that when A = 0 DEMUX 1 is enabled and when A = 1 DEMUX 2 is enabled.

Example 4.15.3 *Implement 1 : 16 demultiplexer using 1 : 4 demultiplexers.*

Solution : The 1 : 16 demultiplexer has 16 outputs. To select one of the 16 output, the circuit needs 4 ($\because 2^4 = 16$) select lines. Each 1 : 4 demultiplexer requires 2 select lines.

Step 1 : Connect two least significant select lines (S_1, S_0) to select lines of four 4 : 1 demultiplexer.

Step 2 : Connect one more 4 : 1 demultiplexer such that its four outputs are routed to the data inputs of the four demultiplexers. Connect higher select lines (S_3, S_2) to the select lines of this demultiplexer.

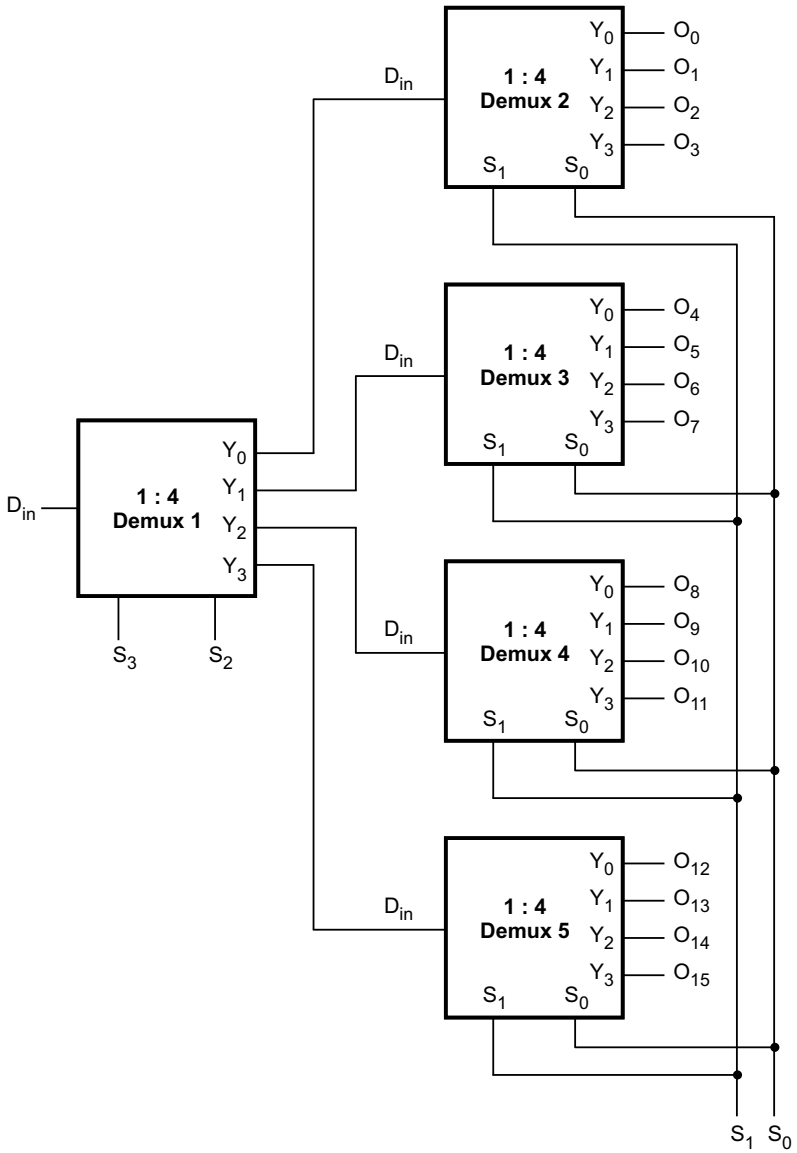


Fig. 4.15.6 1 : 16 Demux using 1 : 4 Demux

Examples for Practice

Example 4.15.4 *Draw 1 : 64 demultiplexer tree using 1 : 16 demultiplexer.*

Example 4.15.5 *Draw 1 : 64 demultiplexer tree using 1 : 8 demultiplexer.*

4.15.3 Implementation of Combinational Logic using Demultiplexer

Example 4.15.6 *Implement full subtractor using demultiplexer.* **Dec.-09, Marks 10**

Solution :

Step 1 : Write the truth table of full subtractor.

Step 2 : Represent output of full-subtractors in minterm form.

For full subtractor difference D function can be written as $D = f(A, B, C) = \sum m(1, 2, 4, 7)$ and B_{out} function can be written as,

$$B_{out} = F(A, B, C) = \sum m(1, 2, 3, 7)$$

Step 3 : Logically OR the outputs corresponding to minterms.

A	B	B_{in}	D	B_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table 4.15.1 Truth table of full subtractor

With D_{in} input 1, demultiplexer gives minterms at the output so by logically ORing required minterms we can implement Boolean functions for full subtractor. Fig. 4.15.7 shows the implementation of full subtractor using demultiplexer.

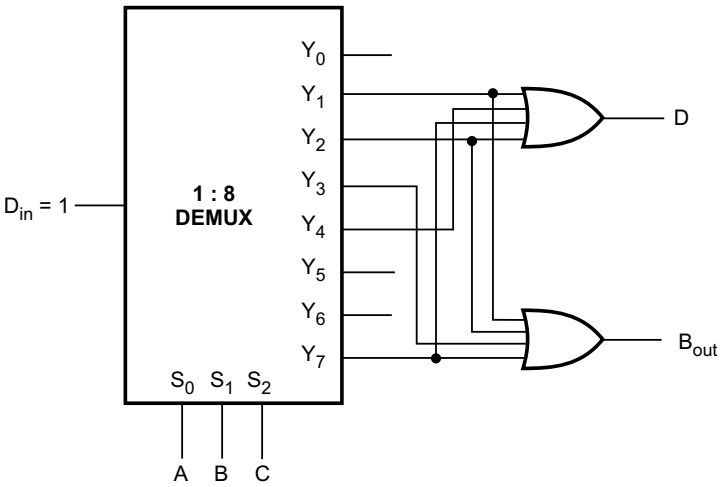


Fig. 4.15.7 Full subtractor using 1 : 8 demultiplexer

Example 4.15.7 *Implement the following functions using demultiplexer :*

$$f_1(A,B,C) = \sum m(0, 3, 7)$$

$$f_2(A,B,C) = \sum m(1, 2, 5).$$

Solution :

f_1 (A, B, C) = $\sum$ m (0, 3, 7)

f_2 (A, B, C) = $\sum$ m (1, 2, 5)

Implementation using 1 : 8 demultiplexer.

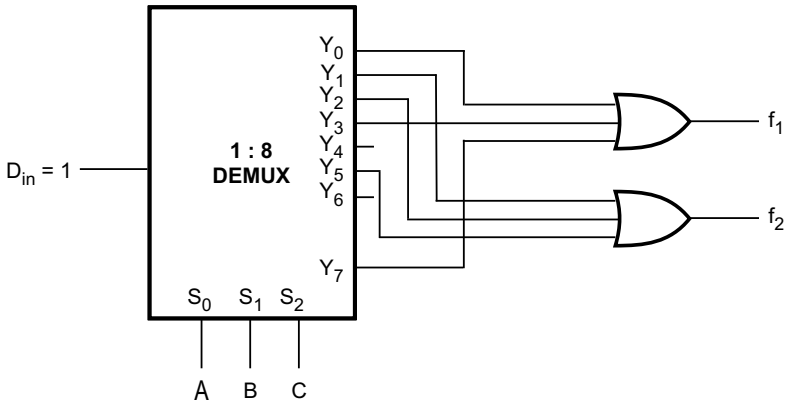


Fig. 4.15.8

Examples for Practice

Example 4.15.8 *Implement full adder using demultiplexer.*

Example 4.15.9 *Implement the following functions using demultiplexer*

$f_1(A, B, C) = \sum m(1, 5, 7)$, $f_2(A, B, C) = \sum m(3, 6, 7)$

4.15.4 Applications of Demultiplexer

1. It can be used as a decoder.
2. It can be used as a data distributer.
3. It is used in time division multiplexing at the receiving end as a data separator.
4. It can be used to implement Boolean expressions.

4.15.5 Demultiplexer ICs

IC Number	Function
74154	1 : 16 Demultiplexer
74155	Dual 1 : 4 Demultiplexer
Refer Appendix A for details	

Table 4.15.2 Demultiplexer ICs

Review Questions

1. *Define demultiplexer.* **May-04, Marks 2**
2. *Differentiate between multiplexer and demultiplexer.*
3. *State the applications of demultiplexer.*

4.16 Decoder

Dec.-06, May-07

A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different.

The Fig. 4.16.1 shows the general structure of the decoder circuit. As shown in the Fig. 4.16.1, the encoded information is presented as n inputs producing 2^n possible outputs. The 2^n output values are from 0 through $2^n - 1$.

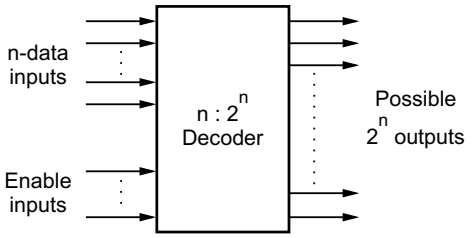


Fig. 4.16.1 General structure of decoder

Usually, a decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

4.16.1 Binary Decoder

A decoder which has an n-bit binary input code and a one activated output out of 2^n output code is called **binary decoder**. A binary decoder is used when it is necessary to activate exactly one of 2^n outputs based on an n-bit input value.

Fig. 4.16.2 shows 2 to 4 decoder. Here, 2 inputs are decoded into four outputs, each output representing one of the minterms of the 2 input variables. The two inverters provide the complement of the inputs, and each one of four AND gates generates one of the minterms.

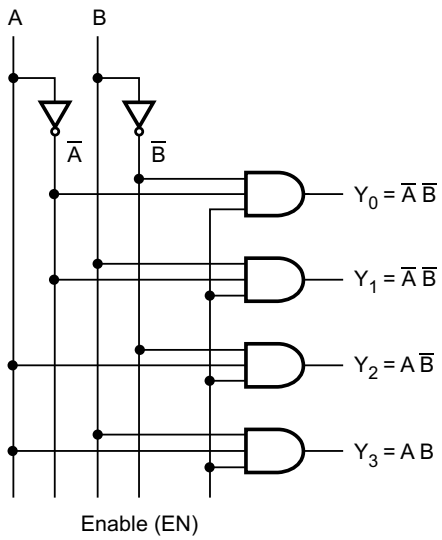


Fig. 4.16.2 2 to 4 line decoder

The Table 4.16.1 shows the truth table for a 2 to 4 decoder. As shown in the truth table, if enable input is 1 (EN = 1), one, and only one, of the outputs Y_0 to Y_3, is active for a given input. The output Y_0 is active, i.e. $Y_0 = 1$ when inputs A = B = 0, the output Y_1 is active when inputs A = 0 and B = 1. If enable input is 0, i.e. EN = 0, then all the outputs are 0.

Inputs			Outputs			
EN	A	B	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 4.16.1 Truth table for a 2 to 4 decoder

Example 4.16.1 *Draw the circuit for 3 to 8 decoder and explain.*

Solution : Fig. 4.16.3 shows 3 to 8 line decoder. Here, 3 inputs are decoded into eight outputs, each output represent one of the minterms of the 3 input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. Enable input is provided to activate decoded output based on data inputs A, B, and C. The table shows the truth table for 3 to 8 decoder.

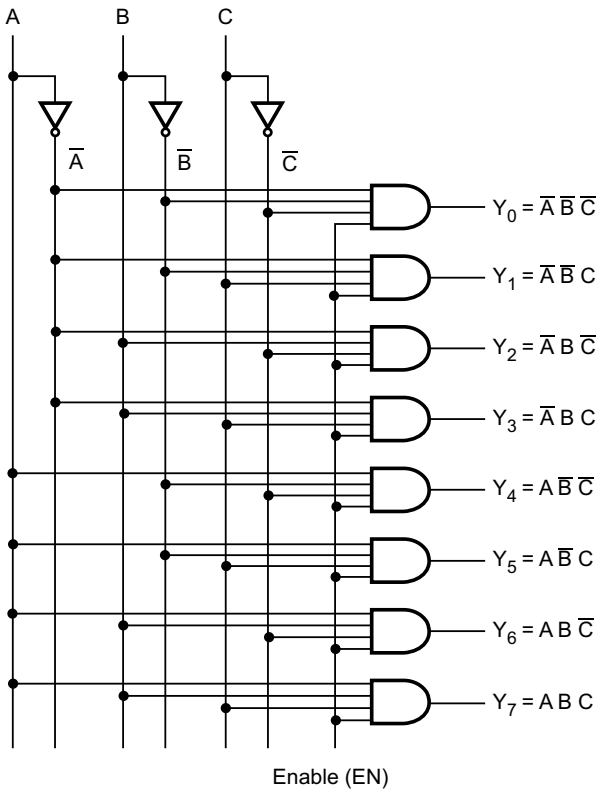


Fig. 4.16.3 3 : 8 line decoder

Inputs				Outputs							
EN	A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Table 4.16.2 Truth table for a 3 to 8 decoder

4.16.2 The 74X138 3-to-8 Decoder

The 74X138 is a commercially available 3-to-8 decoder. It accepts three binary inputs (A, B, C) and when enabled, provides eight individual active low outputs (Y_0 - Y_7). The device has three enable inputs : two active low ($\overline{G}_{2A}, \overline{G}_{2B}$) and one active high ($G_1$). Fig. 4.16.4 and Table 4.16.3 show logic symbol and function table respectively.

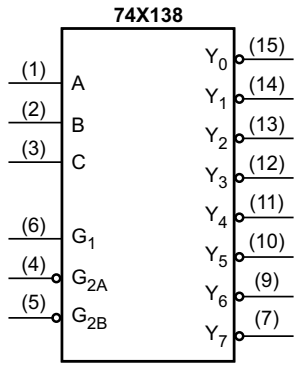


Fig. 4.16.4 Logic symbol

Inputs						Outputs							
G_{2B}	G_{2A}	G_1	C	B	A	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	0	X	X	X	1	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1	1	1	0
0	0	1	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	0	1	1	1	1	1	0	1	1
0	0	1	0	1	1	1	1	1	1	0	1	1	1
0	0	1	1	0	0	1	1	1	0	1	1	1	1
0	0	1	1	0	1	1	1	0	1	1	1	1	1
0	0	1	1	1	0	1	0	1	1	1	1	1	1
0	0	1	1	1	1	0	1	1	1	1	1	1	1

Table 4.16.3 Function table

4.16.3 The 74X139 Dual 2 to 4 Decoder

The 74X139 consists of two independent and identical 2-to-4 decoders. The enable inputs and outputs of IC 74X139 are active low. Fig. 4.16.5 shows the logic symbol and Table 4.16.3 shows the function table for IC 74X139. The Table 4.16.4 shows the truth table for one half of a 74X139 dual 2-to-4 decoder. The truth table for other half is same as first half.

Inputs			Outputs			
$\overline{G}$	B	A	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

Table 4.16.4 Truth table for one half of a 74X139

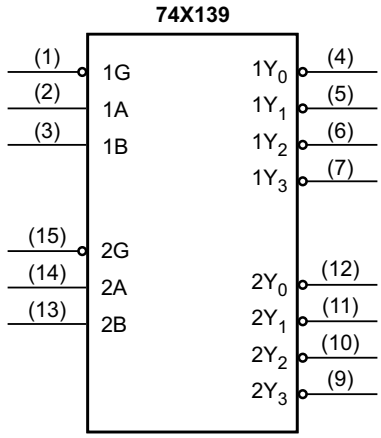


Fig. 4.16.5 Logic symbol for IC 74X139

4.16.4 Expanding Cascading Decoders

Binary decoder circuits can be connected together to form a larger decoder circuit. Fig. 4.16.6 shows the 4 × 16 decoder using two 3 × 8 decoders.

Here, one input line (D) is used to enable/disable the decoders. When D = 0, the top decoder is enabled and the other is disabled. Thus the bottom decoder outputs are all 1s, and the top eight outputs generate minterms 0 0 0 0 to 0 1 1 1. When D=1, the enable conditions are reversed and thus bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 1s.

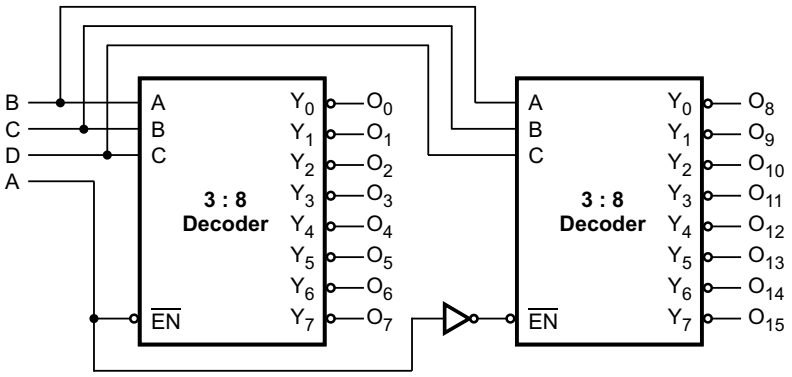


Fig. 4.16.6

Example 4.16.2 *Design 5-to-32 decoder using one 2-to-4 and four 3-to-8 decoder ICs.*

Solution : The Fig. 4.16.7 shows the construction of 5-to-32 decoder using four 74LS138s and half 74LS139. The half section of 74LS139 IC is used as a 2-to-4 decoder to decode the two higher order inputs, D and E. The four outputs of this decoder are used to enable one of the four 3 to 8 decoders. The three lower order inputs A, B and C are connected in parallel to four 3 to 8 decoders. This means that the same output pin of each of the four 3-to-8 decoders is selected but only one is enabled. The remaining

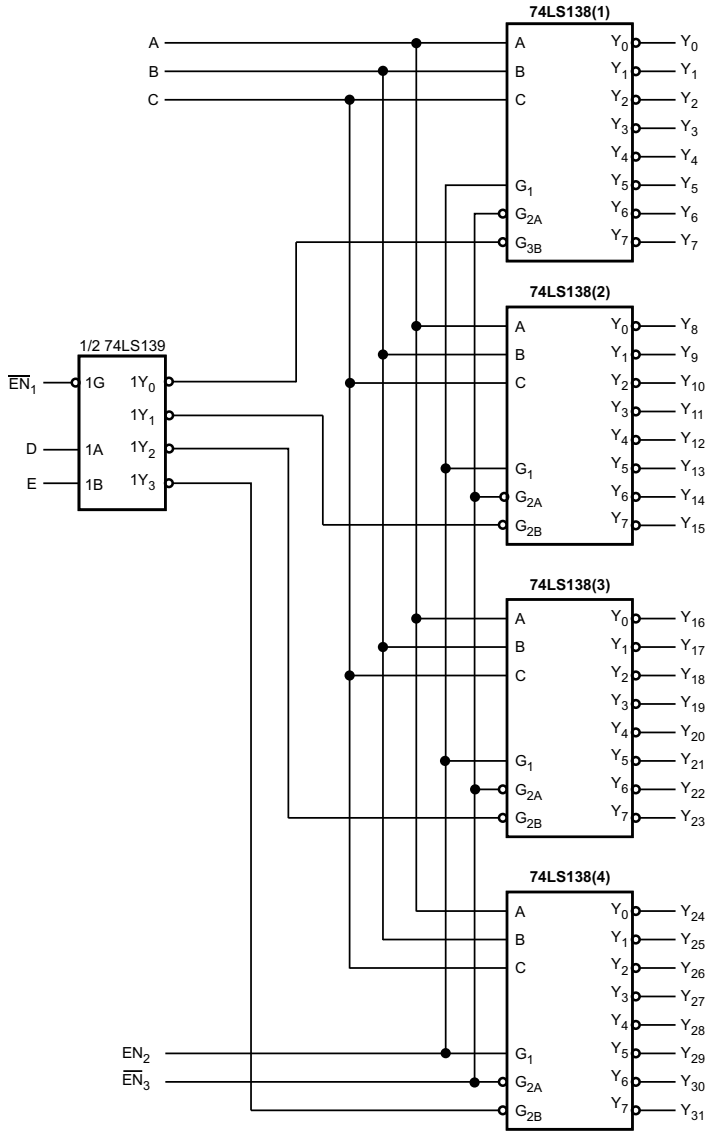


Fig. 4.16.7 5-to-32 decoder using 74LS138 and 74LS139

enable signals of four 3-to-8 decoder ICs are connected in parallel to construct enable signals for 5-to-32 decoder.

4.16.5 Realization of Boolean Function using Decoder

The combination of decoder and external logic gates can be used to implement single or multiple output functions. We know that decoder can have one of the two output states; either active low or active high. Let us see the significance of these output states in the implementation of binary function.

When decoder output is active high, it generates minterms (product terms) for input variables; i.e. it makes selected output logic 1. In such case to implement SOP function we have to take sum of selected product terms generated by decoder.

Examples for Understanding

Example 4.16.3 *Implement Boolean function* $F = \sum m\ (1, 2, 3, 7)$ *using 3 : 8 decoder.*

Solution :

Step 1 : Connect function variables as inputs to the decoder.

Step 2 : Logically OR the outputs correspond to present minterms to obtain the output.

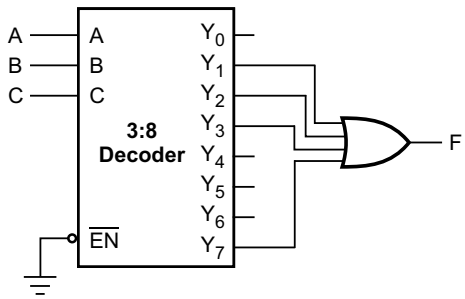


Fig. 4.16.8

Example 4.16.4 *Implement the following multiple output combinational logic using a 4 line to 16 line decoder.*

$Y_1 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,\overline{B}\,C\,\overline{D} + \overline{A}\,B\,C\,\overline{D} + A\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,C\,D$

$Y_2 = \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,\overline{C}\,D + A\,B\,\overline{C}\,D$

$Y_3 = \overline{A}\,B\,C\,D + A\,B\,C\,\overline{D} + ABCD.$

Solution :

Step 1 : Write the function in their minterm forms

$\therefore \quad Y_1 = \sum m\ (0, 3, 2, 6, 10, 11) \qquad Y_2 = \sum m\ (1, 4, 5, 13) \qquad Y_3 = \sum m\ (7, 14, 15)$

Step 2 : Logically OR the outputs of decoder corresponding to the minterms in the functions.

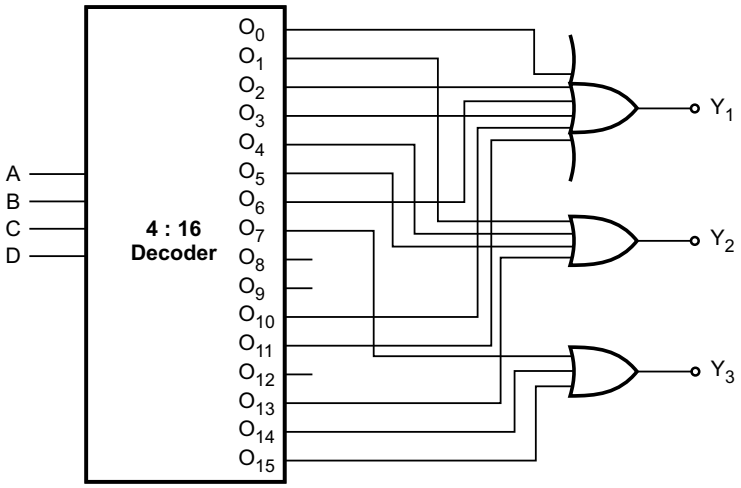


Fig. 4.16.9

Examples with Solutions

Example 4.16.5 *Implement the following multiple output combinational logic circuit using a 4-line to 16-line decoder.*

$f_1 = \Sigma m\ (1, 2, 4, 7, 8, 11, 12, 13)$, $f_2 = \Sigma m\ (2, 3, 9, 11)$

$f_3 = \Sigma m\ (10, 12, 13, 14)$, $f_4 = \Sigma m\ (2, 4, 8)$ **Dec.-08, Marks 10**

Solution :

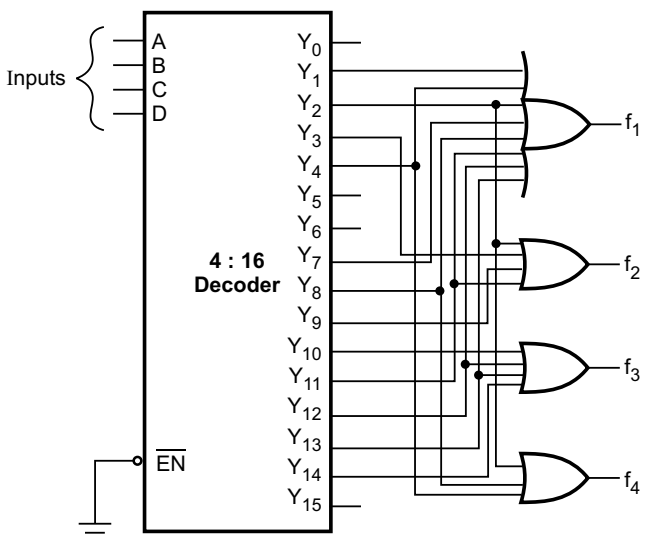


Fig. 4.16.10

Example 4.16.6 *Design and implement a full adder circuit using a 3 : 8 decoder.*

May-11, Marks 5

Solution : Truth table for full adder is as shown in the Table 4.16.5.

Inputs			Outputs	
A	B	C_{in}	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4.16.5 Truth table for full-adder

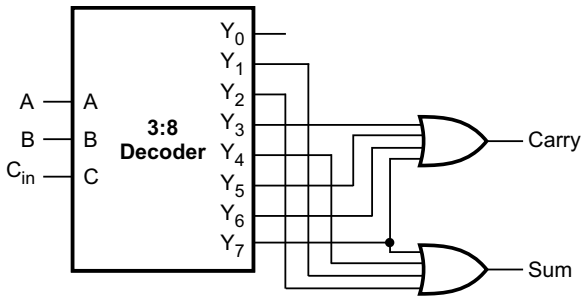


Fig. 4.16.11

Example 4.16.7 *Design an excess-3 to BCD code converter using decoder and gates.*

Solution : The truth table for excess-3 to BCD code converter is as shown in the Table 4.16.6.

E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

Table 4.16.6

Realization

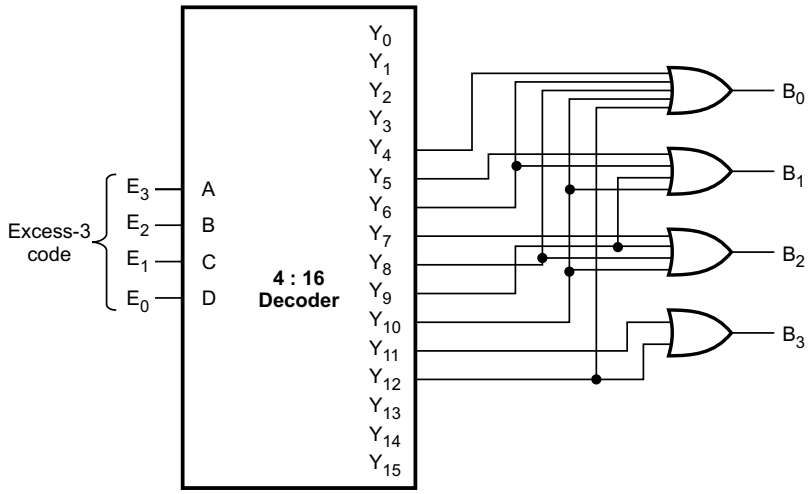


Fig. 4.16.12

Examples for Practice

Example 4.16.8 *Implement the following Boolean functions using decoder and OR gates :*

$$F_1(A,B,C,D) = \sum (2,4,7,9)$$

$$F_2(A,B,C,D) = \sum (10,13,14,15)$$

Example 4.16.9 *Implement the logic circuit for full subtractor using decoder.*

4.16.6 Applications of Decoder

The uses of decoders are :

- Code converters
- Implementation of combinational circuits
- Address decoding
- BCD to 7-segment decoder

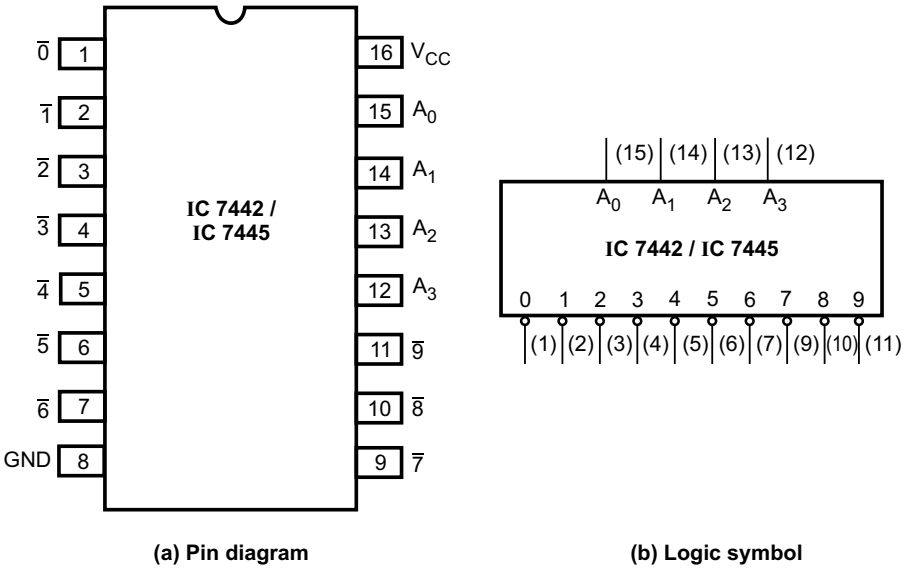
4.16.7 Decoder ICs

IC Number	Function
74138	3 : 8 Decoder
74139	Dual 2 : 4 Decoder
7442	BCD to decimal decoder
7447	BCD to 7-segment decoder
Refer Appendix A for details	

Table 4.16.7 Decoder ICs

4.16.8 BCD to Decimal Decoder

BCD decoders have four inputs and 10 outputs. The four-bit BCD input is decoded to activate one of the ten outputs. The 74XX42 is a BCD to decimal decoder. It accepts four active high BCD inputs and provides 10 independent active low outputs. Fig. 4.16.13 (a) shows pin diagram and logic symbol for IC 7442. The active low outputs of IC 7442 facilitates addressing other MSI units with active low input enables.



(a) Pin diagram (b) Logic symbol

Fig. 4.16.13

The logic design of 7442 ensures that all outputs are high when binary codes greater than nine are applied to the inputs. The most significant input, A_3 produces a useful inhibit function when the 7442 is used as a 3 to 8 decoder.

The IC 7445 is also a BCD to decimal decoder with identical pin description as that of IC 7442. However, IC 7445 provides open collector outputs.

4.16.9 BCD to Seven Segment Decoder

In most practical applications, seven segment displays are used to give a visual indication of the output states of digital ICs such as decade counters, latches etc. These outputs are usually in four bit BCD (binary coded decimal) form, and are thus not suitable for directly driving seven segment displays. The special BCD to seven segment decoder/driver ICs are used to convert the BCD signal into a form suitable for driving these displays. In this sections, we are going to study LED and LCD decoders/drivers for seven segment displays. Let us tabulate the segments activated during each digit display.

Digit	Segments Activated	Display
0	a, b, c, d, e, f	
1	b, c	
2	a, b, d, e, g	
3	a, b, c, d, g	
4	b, c, f, g	
5	a, c, d, f, g	
6	a, c, d, e, f, g	
7	a, b, c	
8	a, b, c, d, e, f, g	
9	a, b, c, d, f, g	

Table 4.16.8

From the Table 4.16.8 we can determine the truth table for BCD-to-7 segment decoder/driver. This truth table also depends on the construction of 7-segment display. If 7-segment display is common anode, the segment driver output must be active low to glow the segment. In case of common cathode type 7-segment display, the segment driver output must be active high to glow the segment. Table 4.16.9 and 4.16.10 show the truth tables for both BCD to 7 segment decoder/driver with common cathode display and with common anode display respectively.

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Table 4.16.9 Truth table for BCD-to-common cathode 7-segment decoder/driver

Digit	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	0	0	1	1	0
4	0	1	0	0	1	0	0	1	1	0	0
5	0	1	0	1	0	1	0	0	1	0	0
6	0	1	1	0	0	1	0	0	0	0	0
7	0	1	1	1	0	0	0	1	1	1	1
8	1	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	0	0	1	0	0

Table 4.16.10 Truth table for BCD-to-common anode 7-segment decoder/driver

Let us design the combinational circuit for common cathode 7-segment display/driver.

K-map simplification

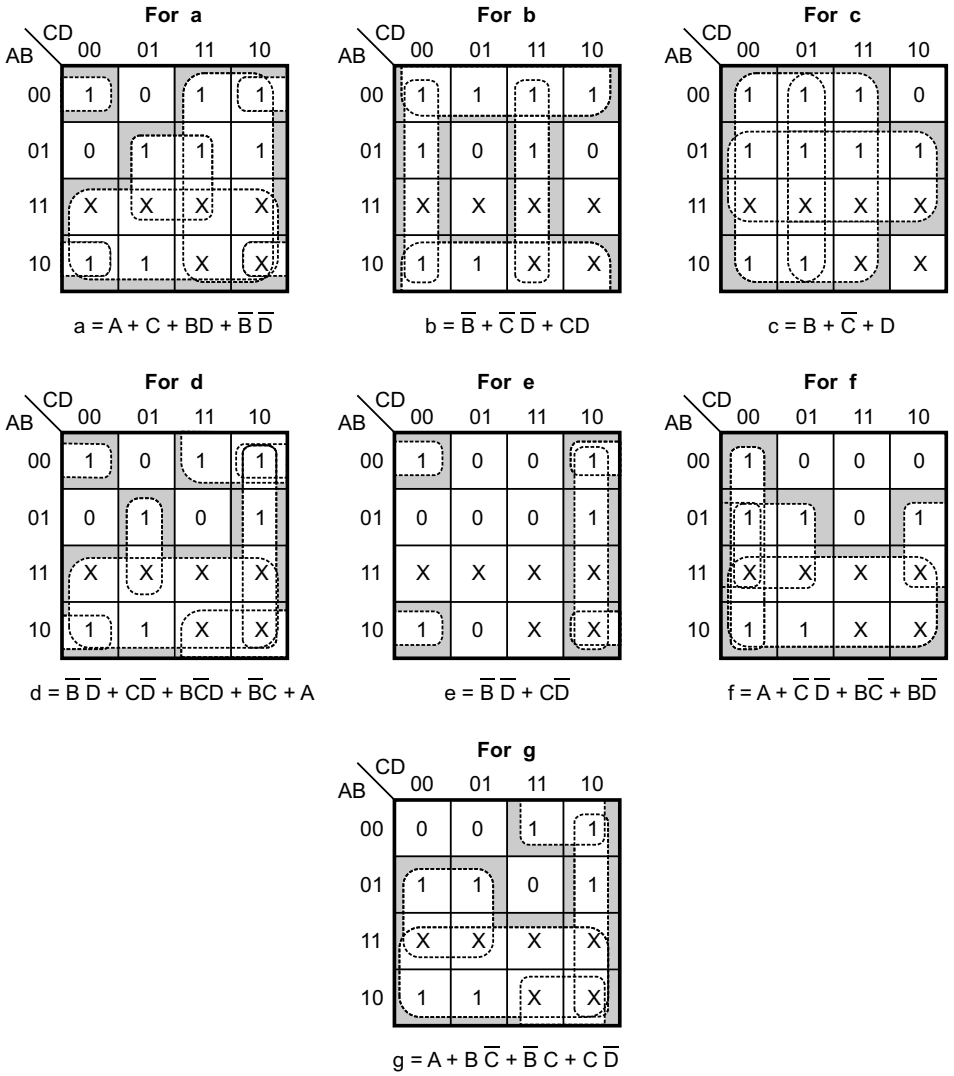

For a
CD
AB
00 01 11 10
00 1 0 1 1
01 0 1 1 1
11 X X X X
10 1 1 X X
a = A + C + BD + $\overline{B}\,\overline{D}$
For b
CD
AB
00 01 11 10
00 1 1 1 1
01 1 0 1 0
11 X X X X
10 1 1 X X
b = $\overline{B}$ + $\overline{C}\,\overline{D}$ + CD
For c
CD
AB
00 01 11 10
00 1 1 1 0
01 1 1 1 1
11 X X X X
10 1 1 X X
c = B + $\overline{C}$ + D
For d
CD
AB
00 01 11 10
00 1 0 1 1
01 0 1 0 1
11 X X X X
10 1 1 X X
d = $\overline{B}\,\overline{D}$ + C$\overline{D}$ + B$\overline{C}$D + $\overline{B}$C + A
For e
CD
AB
00 01 11 10
00 1 0 0 1
01 0 0 0 1
11 X X X X
10 1 0 X X
e = $\overline{B}\,\overline{D}$ + C$\overline{D}$
For f
CD
AB
00 01 11 10
00 1 0 0 0
01 1 1 0 1
11 X X X X
10 1 1 X X
f = A + $\overline{C}\,\overline{D}$ + B$\overline{C}$ + B$\overline{D}$
For g
CD
AB
00 01 11 10
00 0 0 1 1
01 1 1 0 1
11 X X X X
10 1 1 X X
g = A + B $\overline{C}$ + $\overline{B}$ C + C $\overline{D}$

Fig. 4.16.14

Logic diagram

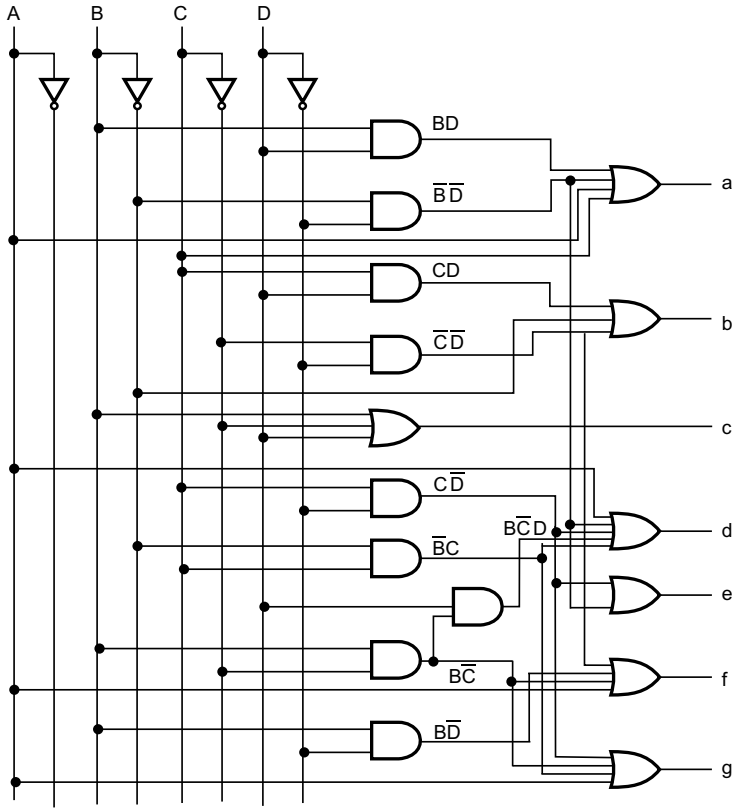


Fig. 4.16.15

Example 4.16.10 *Implement BCD-to-7 segment decoder for common anode using 4 : 16 decoder.*

Solution : Referring the truth table from Table 4.16.9 we can implement BCD-to-7 segment decoder for common anode as shown in the Fig. 4.16.16. (See Fig. 4.16.16 shown on next page).

Review Questions

1. *Define decoder.*
2. *Define binary decoder.*
3. *Explain the working of 2 : 4 binary decoder.*
4. *Draw a 4 × 16 decoder constructed with two 3 × 8 decoders.* **May-07, Marks 2**
5. *State the procedure to implement Boolean function using decoder.*
6. *Mention the uses of decoders.* **Dec.-06, Marks 2**
7. *Write a note on BCD to decimal decoder.*
8. *Write a note on BCD to 7-segment decoder.*

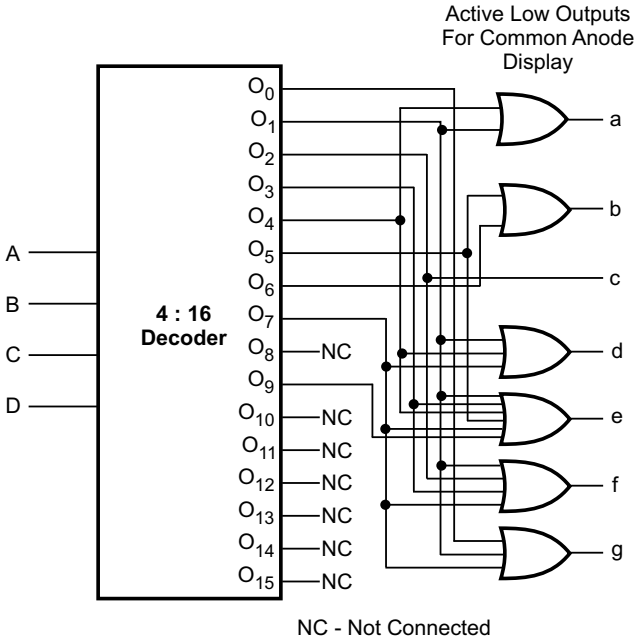


Fig. 4.16.16 BCD-to-7 segment decoder using 4 : 16 decoder

4.17 Encoder

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. In encoder the output lines generate the binary code corresponding to the input value. The Fig. 4.17.1 shows the general structure of the encoder circuit. As shown in the Fig. 4.17.1 the decoded information is presented as 2^n inputs producing n possible outputs.

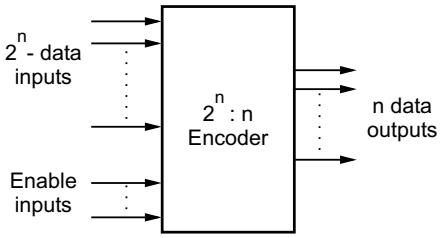


Fig. 4.17.1 General structure of encoder

4.17.1 Decimal to BCD Encoder

The decimal to BCD encoder, usually has ten input lines and four output lines. The decoded decimal data acts as an input for encoder and encoded BCD output is available on the four output lines.

The Fig. 4.17.2 shows the logic symbol for decimal to BCD encoder IC, IC 74XX147. It has nine input lines and four output lines. Both input and output lines are asserted active low. It is important to note that there is no input line for decimal zero. When this condition occurs, all output lines are 1. The function table for the 74XX147 is shown in Table 4.17.1.

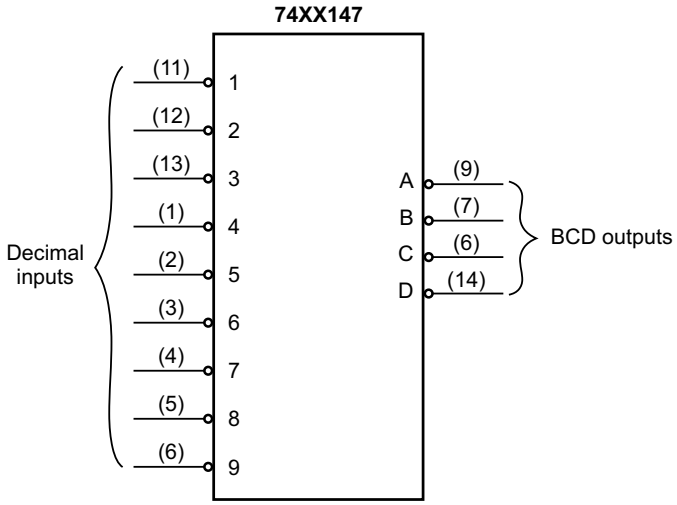


Fig. 4.17.2 Logic symbol for 74XX147 (Decimal to BCD encoder)

Decimal value	Inputs									Outputs			
	1	2	3	4	5	6	7	8	9	D	C	B	A
0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0
2	x	0	1	1	1	1	1	1	1	1	1	0	1
3	x	x	0	1	1	1	1	1	1	1	1	0	0
4	x	x	x	0	1	1	1	1	1	1	0	1	1
5	x	x	x	x	0	1	1	1	1	1	0	1	0
6	x	x	x	x	x	0	1	1	1	1	0	0	1
7	x	x	x	x	x	x	0	1	1	1	0	0	0
8	x	x	x	x	x	x	x	0	1	0	1	1	1
9	x	x	x	x	x	x	x	x	0	0	1	1	0

x indicates don't care condition

Table 4.17.1 Truth table for decimal to BCD encoder

4.17.2 Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Table 4.17.2 shows truth table of 4-bit priority encoder.

Table 4.17.2 shows D_3 input with highest priority and D_0 input with lowest priority. When D_3 input is high, regardless of other inputs output is 11. The D_2 has the next priority. Thus, when D_3 = 0 and D_2 = 1, regardless of other two lower priority input, output is 10. The output for D_1 is generated only if higher priority inputs are 0, and so on. The output V (a valid output indicator) indicates, one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs (Y_1 and Y_0) of the circuit are not used.

Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 4.17.2 Truth table of 4-bit priority encoder

K-map simplification

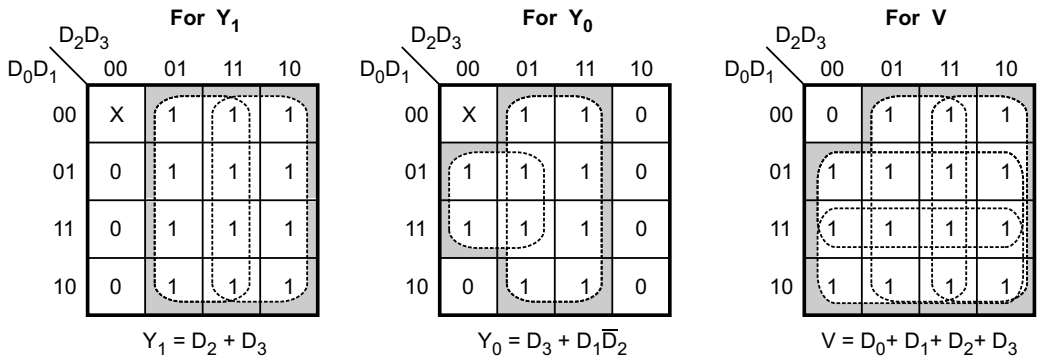


Fig. 4.17.3

Logic diagram

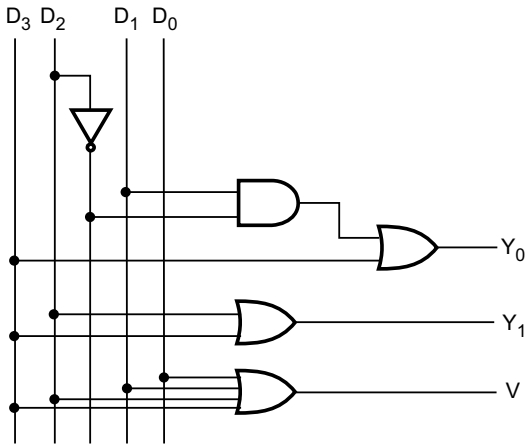


Fig. 4.17.4

4.17.3 Octal to Binary Encoder

Fig. 4.17.5 shows octal to binary encoder. It has eight inputs, one for each octal digit, and three outputs that generate the corresponding binary code. In encoders it is assumed that only one input has a value of 1 at any given time; otherwise the circuit is meaningless.

Table 4.17.3 shows the truth table of octal to binary converter.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Table 4.17.3 Truth table of octal-to-binary encoder

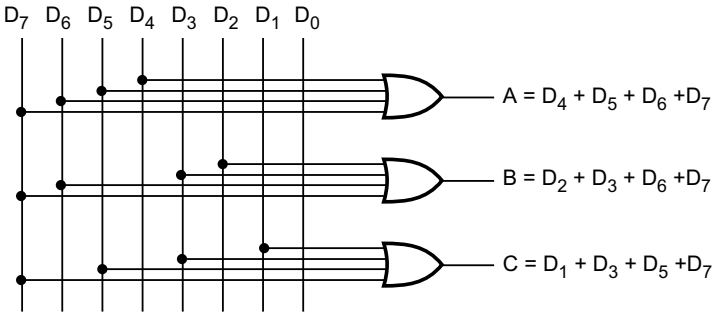


Fig. 4.17.5 Octal to binary encoder

The above circuit has one more ambiguity that when all inputs are 0s the outputs are 0s. The zero output can also be generated when $D_0 = 1$. This ambiguity can be resolved by providing an additional output that specifies the valid condition.

4.17.4 Encoder ICs

IC Number	Function
74147	Decimal to BCD encoder
74148	8-input priority Encoder
Note : Refer Appendix A for details	

Table 4.17.4 Encoder ICs

Review Questions

1. *Define encoder.*
2. *Write a note on encoders.*
3. *Explain decimal to BCD encoder with logic symbol and truth table.*
4. *Define priority encoder.*
5. *Explain the octal to binary encoder.*

4.18 Parity Generators and Checkers

Dec.-08, 12, May-10

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message, including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker. In this section, we will study the design of parity generator and parity checker.

In even parity the added parity bit will make the total number of 1s an even amount. In odd parity the added parity bit will make the total number of 1s an odd amount. Table 4.18.1 shows the 3-bit message with even parity and odd parity.

3-bit Message			Odd Parity Bit	Even Parity Bit
A	B	C		
0	0	0	1	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Table 4.18.1 Parity generator truth table for even and odd parity

Let us design the even parity generator.

K-map simplification

$$
\begin{aligned}
P &= \overline{A}\,\overline{B}\,C + \overline{A}\,B\,\overline{C} + A\,\overline{B}\,\overline{C} + A\,B\,C \\
&= \overline{A}\,(\overline{B}\,C + B\,\overline{C}) + A\,(\overline{B}\overline{C} + B\,C) \\
&= \overline{A}\,(B \oplus C) + A\,(B \odot C) \\
&= \overline{A}\,(B \oplus C) + A\,(\overline{B \oplus C}) \\
&= A \oplus (B \oplus C)
\end{aligned}
$$

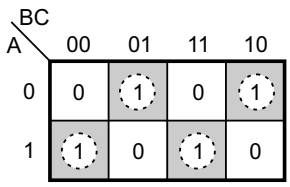


Fig. 4.18.1 (a)

Logic diagram

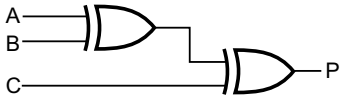


Fig. 4.18.1 (b)

The three bits in the message together with the parity bit are transmitted to their destination, where they are applied to the parity checker circuit. The parity checker circuit checks for possible errors in the transmission. Since the information was transmitted with even parity, the four bits received must have an even number of 1s. An error occurs during the transmission if the four bits received have an odd number of 1s, indicating that one bit has changed in value during transmission. The output of the parity checker, is denoted by PEC (Parity Error Check). It will be equal to 1, if an error occurs, that is, if the four bits received have an odd number of 1s. The table shows the truth table for the even parity checker.

Four Bits Received				Parity Error Check
A	**B**	**C**	**D**	**PEC**
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table 4.18.2 Truth table for even parity checker

K-map simplification

$$PEC = \overline{A}\,\overline{B}\,(\overline{C}D + C\overline{D}) + \overline{A}\,B\,(\overline{C}\,\overline{D} + CD)$$
$$+ A\,B\,(\overline{C}D + C\overline{D}) + A\,\overline{B}\,(\overline{C}\,\overline{D} + CD)$$

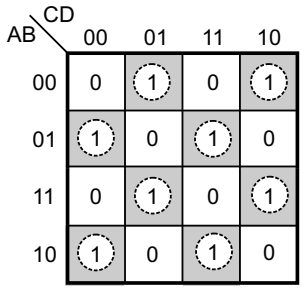
$$= \overline{A}\,\overline{B}\,(C \oplus D) + \overline{A}\,B\,(\overline{C \oplus D})$$

$$+ A\,B\,(C \oplus D) + A\,\overline{B}\,(\overline{C \oplus D})$$

$$= (\overline{A}\,\overline{B} + A\,B)\,(C \oplus D) + (\overline{A}\,B + A\,\overline{B})\,(\overline{C \oplus D})$$

$$= (\overline{A \oplus B})\,(C \oplus D) + (A \oplus B)\,(\overline{C \oplus D})$$

$$= (A \oplus B) \oplus (C \oplus D)$$



PEC = (A⊕B)⊕(C⊕D)

Fig. 4.18.2 (a)

Logic diagram

Let us study the standard IC for parity generator and checker.

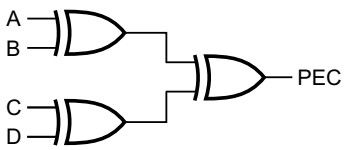
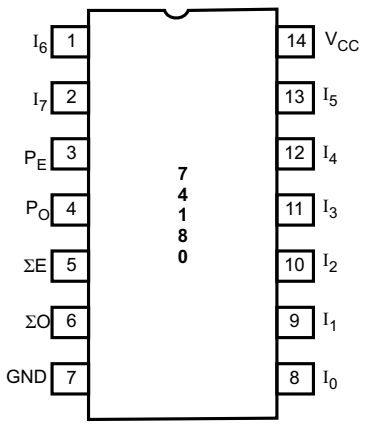


Fig. 4.18.2 (b)

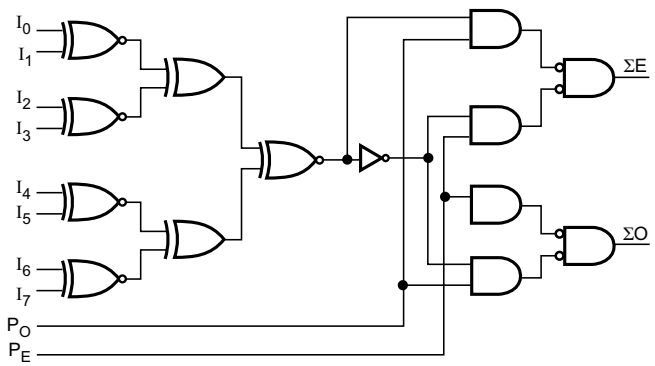
IC 74180 : Parity Generator/Checker

The 74180 is a 9-bit parity generator or checker commonly used to detect errors in high speed data transmission or data retrieval systems. Both **even** and **odd** parity enable inputs and parity outputs are available for generating or checking parity on 8-bits.

The Fig. 4.18.3 (a, b) shows pin diagram and logic diagram for 74180.



(a) Pin diagram



(b) Logic diagram

Fig. 4.18.3

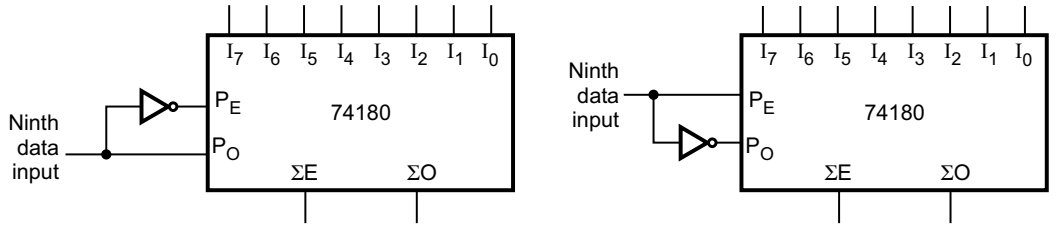
Table 4.18.3 shows the function table for 74180

Inputs			Outputs	
Number of HIGH data Inputs (I_0 - I_7)	P_E	P_O	ΣE	ΣO
Even	1	0	1	0
Odd	1	0	0	1
Even	0	1	0	1
Odd	0	1	1	0
X	1	1	0	0
X	0	0	1	1

Table 4.18.3 Function table for 74180

As shown in the Table 4.18.3, true active-HIGH or true active-LOW parity can be generated at both the even or odd outputs. True active-HIGH parity is established with Even parity enable input (P_E) set HIGH and the Odd parity enable input (P_O) set LOW. True active LOW parity is established when P_E is LOW and P_O is HIGH. When both are enable inputs are at the same logic level, both outputs will be forced to the opposite logic level.

Parity checking of a 9-bit word (8-bits plus parity) is possible by using the two enable inputs plus an inverter as the ninth data input. To check for true active-HIGH parity, the ninth data input is tied to the P_O input and an inverter is connected between the P_O and P_E inputs as shown in the Fig. 4.18.4 (a). To check for true active-LOW parity, the ninth data input is tied to the P_E input and an inverter is connected between the P_E and P_O inputs, as shown in the Fig. 4.18.4 (b).



(a) Check for true active-HIGH parity **(b) Check for true active-LOW parity**

Fig. 4.18.4

Expansion to larger word sizes is accomplished by serially cascading the 74180 in 8-bit increments. The even and odd parity outputs of the first stages are connected to the corresponding P_E and P_O inputs, respectively, of the succeeding stage, as shown in the Fig. 4.18.5.

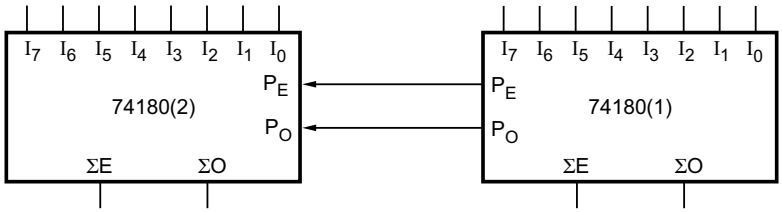


Fig. 4.18.5 Expansion to the larger word

Review Questions

1. *Explain even parity checker.* **Dec.-08, Marks 6**
2. *Draw the circuits diagram for 4 bit odd parity generator.* **May-10, Marks 2**
3. *Design a three bit even parity generator.* **Dec.-12, Marks 10**

4.19 Code Converters

Dec.-03, 04, 09, 12, May-05, 09, 10

There is a wide variety of binary codes used in digital systems. Some of these codes are binary-coded-decimal (BCD), Excess-3, Gray and so on. Many times it is required to convert one code to another.

Let us see the procedure to design code converters :

Step 1 : Write the truth table showing the relationship between input code and output code.

Step 2 : For each output code bit determine the simplified Boolean expression using K-map.

Step 3 : Realize the code converter using logic gates.

Examples for Understanding

Example 4.19.1 *Design a 4-bit binary to BCD converter.*

Step 1 : Form the truth table relating binary and BCD code.

Input code : Binary code : $B_3\ B_2\ B_1\ B_0$ (B_0 LSB)

Output code : BCD (Decimal) code : $D_4\ D_3\ D_2\ D_1\ D_0$ (D_0 LSB)

Binary code				BCD code				
B_3	B_2	B_1	B_0	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

Table 4.19.1 Truth table for binary to BCD conversion

Step 2 : K-map simplification for each BCD output

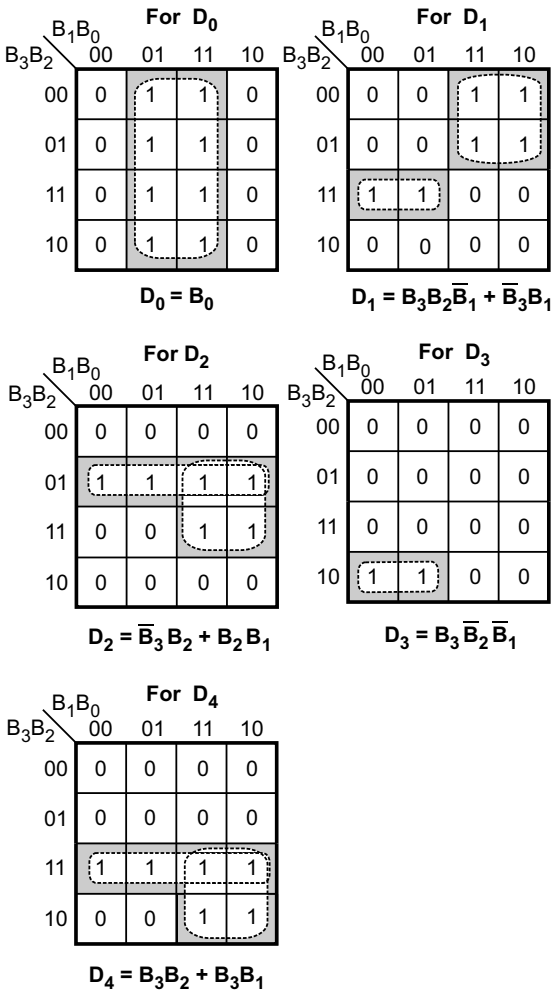


Fig. 4.19.1

Step 3 : Realization of code converter

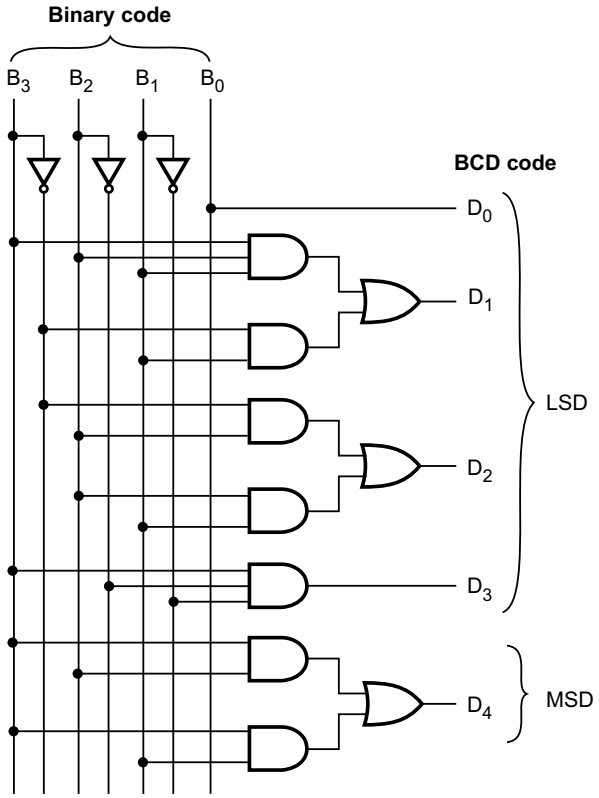


Fig. 4.19.2 Binary to BCD converter

Example 4.19.2 *Design a logic circuit to convert the 8421 BCD to Excess-3 code.*

Dec.-03, 12, May-10, Marks 12

Solution :

Step 1 : Form the truth table relating BCD and Excess-3 code

Excess-3 code is a modified form of a BCD number. The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get Excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as shown in Table 4.19.2.

Input code : BCD code : D_3 D_2 D_1 D_0 (D_0 LSB)

Output code : Excess-3 code : E_3 E_2 E_1 E_0 (E_0 LSB)

Decimal	D_3	D_2	D_1	D_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 4.19.2

Step 2 : K-map simplification for each Excess-3 code output.

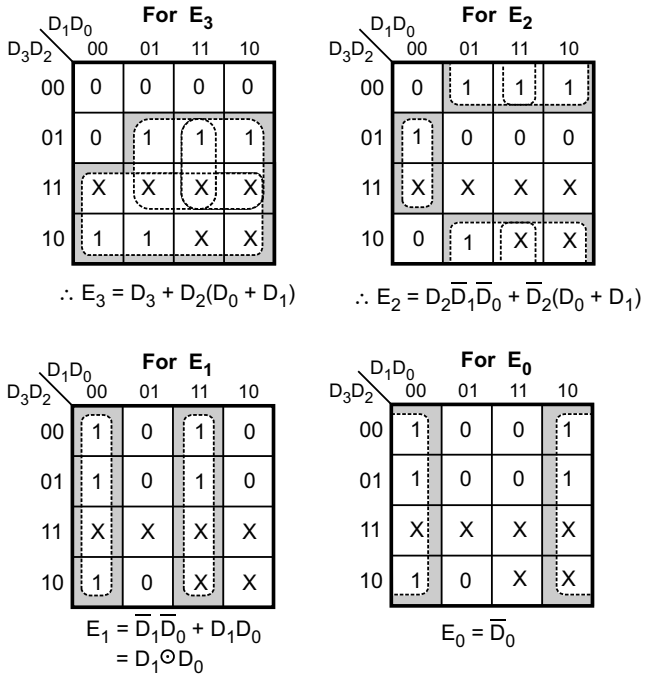


Fig. 4.19.3

Step 3 : Realization of code converter.

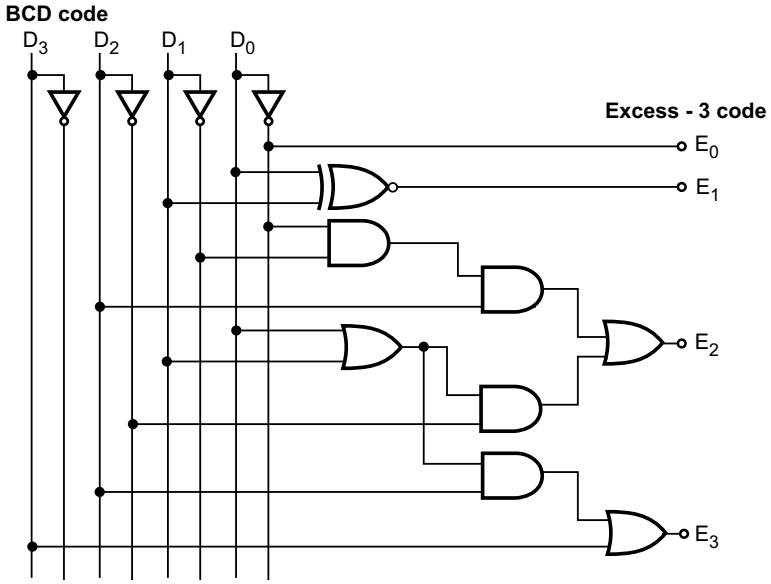


Fig. 4.19.4 BCD to excess-3 code converter

Example 4.19.3 *Design a logic circuit to convert BCD to gray code.* **Dec.-04, Marks 16**

Solution :

Step 1 : Form the truth table relating BCD and gray code.

Input code : BCD code : $D_3\ D_2\ D_1\ D_0$ (D_0 LSB)

Output code : Gray code : $G_3\ G_2\ G_1\ G_0$ (G_0 LSB)

Table 4.19.3 shows truth table for BCD to gray code converter.

BCD code				Gray code			
D_3	D_2	D_1	D_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

Table 4.19.3 Truth table for BCD to gray code converter

Step 2 : K-map simplification

For G_0

D_3D_2 \ D_1D_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$G_0 = \overline{D}_1D_0 + D_1\overline{D}_0$
$= D_1 \oplus D_0$

For G_1

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	x	x	x	x
10	0	0	x	x

$G_1 = D_2\overline{D}_1 + \overline{D}_2D_1$
$= D_2 \oplus D_1$

For G_2

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$G_2 = D_2 + D_3$

For G_3

D_3D_2 \ D_1D_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$G_3 = D_3$

Fig. 4.19.5

Step 3 : Realization of code converter

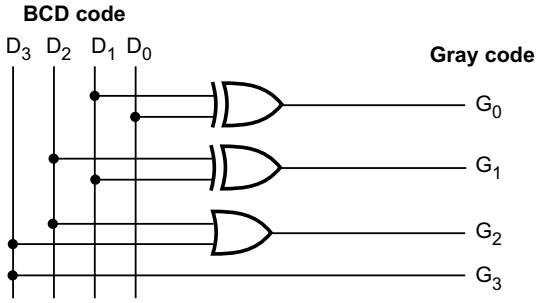


Fig. 4.19.6 BCD to gray code converter

Example 4.19.4 *Design and implement a 8421 to Gray code converter. Realize the converter using only NAND gates.* **May-09, 10, Marks 16**

Solution :

Step 1 : Form the Truth table relating 8421 binary code and Gray code

Input code : Binary code : $B_3\ B_2\ B_1\ B_0$

Output code : Gray code : $G_3\ G_2\ G_1\ G_0$

Decimal	Binary code				Gray code			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

Table 4.19.4

Step 2 : K-map simplification for each gray code output

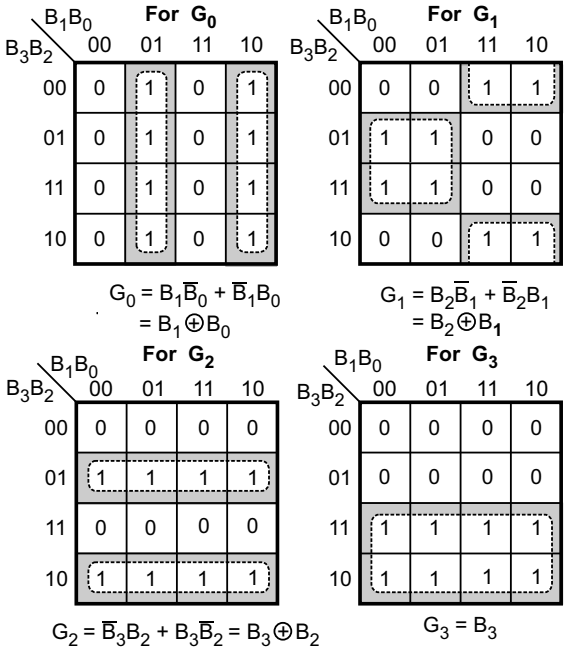


Fig. 4.19.7

Step 3 : Realization of code converter using XOR-gates

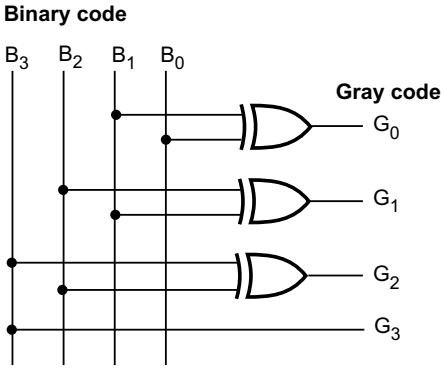


Fig. 4.19.8 Binary to gray code converter

Step 4 : Realization of code converter using NAND gates

For this converter we have derived the Boolean expressions for each gray code output in the sum of product (SOP) form. We can implement SOP expression using AND-OR logic or NAND-NAND logic. Let us see the implementation of code converter using NAND-NAND logic.

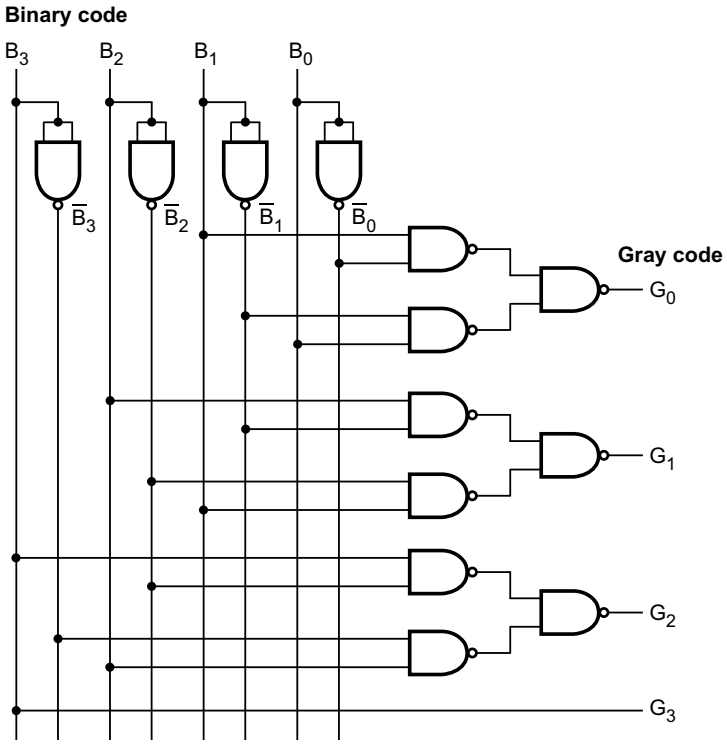


Fig. 4.19.9 Binary to Gray code converter

Example 4.19.5 *Design a Gray to BCD code converter.* **May-05, Marks 16**

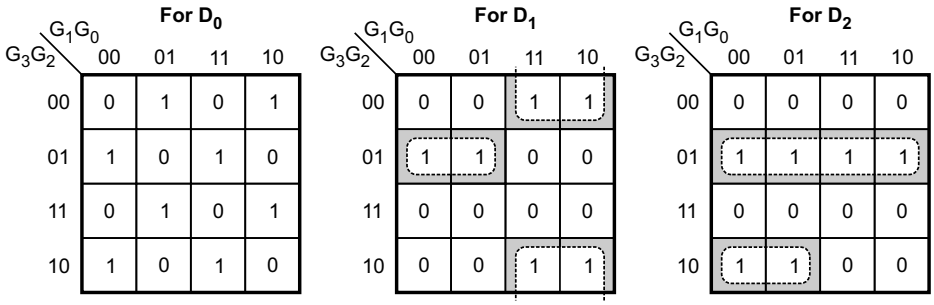
Solution : The Table 4.19.5 shows truth table for gray to BCD code converter.

Gray code				BCD code				
G_3	G_2	G_1	G_0	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	0	1

0	1	0	1	0	0	1	1	0
0	1	0	0	0	0	1	1	1
1	1	0	0	0	1	0	0	0
1	1	0	1	0	1	0	0	1
1	1	1	1	1	0	0	0	0
1	1	1	0	1	0	0	0	1
1	0	1	0	1	0	0	1	0
1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	0	0
1	0	0	0	1	0	1	0	1

Table 4.19.5 Gray to BCD code converter

Kmap-simplification



$D_0 = \overline{G}_3\overline{G}_2\overline{G}_1G_0 + \overline{G}_3\overline{G}_2G_1\overline{G}_0 + \overline{G}_3G_2\overline{G}_1\overline{G}_0 + \overline{G}_3G_2G_1G_0 + G_3\overline{G}_2\overline{G}_1G_0 + G_3\overline{G}_2G_1\overline{G}_0 + G_3\overline{G}_2\overline{G}_1\overline{G}_0 + G_3\overline{G}_2G_1G_0$

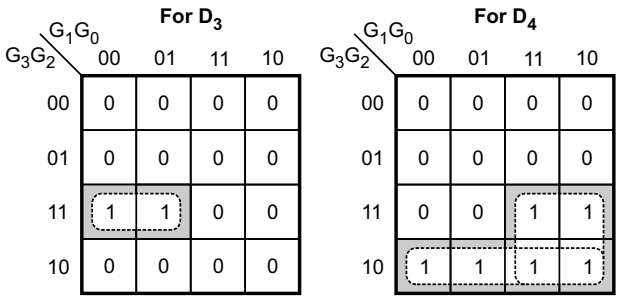
$= \overline{G}_3\overline{G}_2(G_1 \oplus G_0) + \overline{G}_3G_2(\overline{G_1 \oplus G_0}) + G_3G_2(G_1 \oplus G_0) + G_3\overline{G}_2(\overline{G_1 \oplus G_0})$

$= (\overline{G_3 \oplus G_2})(G_1 \oplus G_0) + (G_3 \oplus G_2)(\overline{G_1 \oplus G_0})$

$= G_3 \oplus G_2 \oplus G_1 \oplus G_0$

$D_1 = \overline{G}_3G_2\overline{G}_1 + \overline{G}_2G_1$

$D_2 = \overline{G}_3G_2 + G_3\overline{G}_2\overline{G}_1$



$D_3 = G_3G_2\overline{G}_1$

$D_4 = G_3\overline{G}_2 + G_3G_1$

Fig. 4.19.10

Logic Diagram

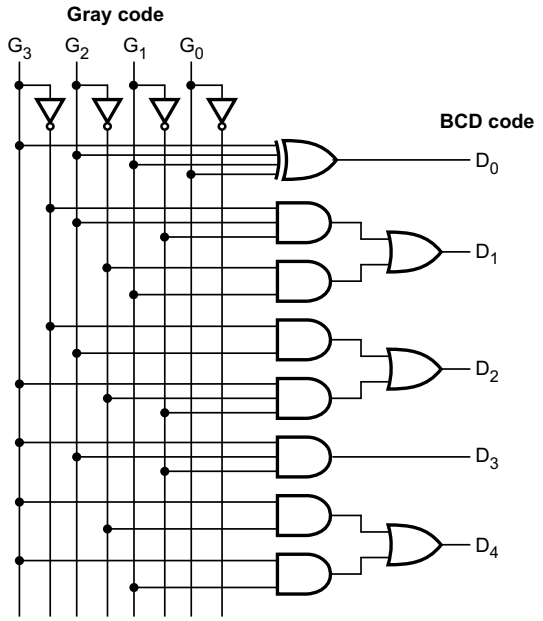


Fig. 4.19.11

Examples for Practice

Example 4.19.6 *Design a logic circuit to convert excess-3 code to BCD code.*

(Ans. : $D_0 = \overline{E}_0$, $D_1 = E_1 \oplus E_0$
$D_2 = \overline{E}_2\,\overline{E}_1 + E_2\,E_1\,E_0 + E_3\,E_1\,\overline{E}_0$ $D_3 = E_3\,E_2 + E_3\,E_1\,E_0$**)**

Example 4.19.7 *Design a logic circuit to convert gray code to binary code.*

Dec.-09, Marks 16

(Ans. : $B_0 = G_3 \oplus G_2 \oplus G_1 \oplus G_0$ $B_1 = G_3 \oplus G_2 \oplus G_1$
$B_2 = G_3 \oplus G_2$ $B_3 = G_3$**)**

4.20 Magnitude Comparator

May-08, 11, 12

A comparator is a special combinational circuit designed primarily to compare the relative magnitude of two binary numbers. Fig. 4.20.1 shows the block diagram of an n-bit comparator. It receives two n-bit numbers A and B as inputs and the outputs are A > B, A = B and A < B. Depending upon the relative magnitudes of the two number, one of the outputs will be high.

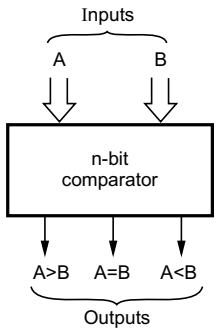


Fig. 4.20.1 Block diagram of n-bit comparator

Example 4.20.1 *Design 2-bit comparator using gates.*

Solution : The truth table for 2-bit is given in Table 4.20.1.

Inputs				Outputs		
A_1	A_0	B_1	B_0	A > B	A = B	A < B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Table 4.20.1

K-map simplification

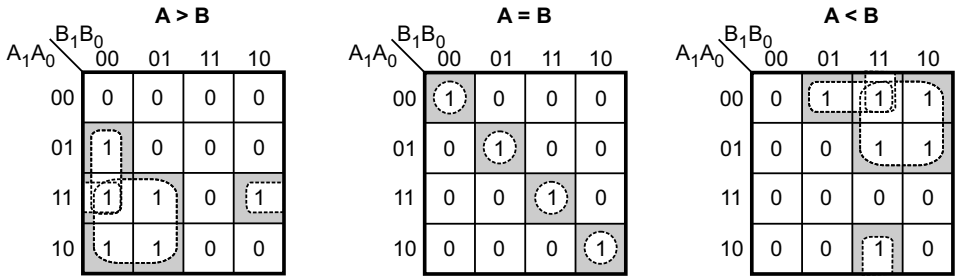


Fig. 4.20.2

$$(A = B) = \overline{A}_1 \overline{A}_0 \overline{B}_1\overline{B}_0 + \overline{A}_1 A_0 \overline{B}_1 B_0 + A_1 A_0 B_1 B_0 + A_1 \overline{A}_0 B_1 \overline{B}_0$$

$$= \overline{A}_1\overline{B}_1 (\overline{A}_0\overline{B}_0 + A_0 B_0) + A_1 B_1 (A_0 B_0 + \overline{A}_0\overline{B}_0)$$

$$= (A_0 \odot B_0) (A_1 \odot B_1)$$

$$(A < B) = \overline{A}_1\overline{A}_0 B_0 + \overline{A}_0 B_1 B_0 + \overline{A}_1 B_1$$

Logic diagram

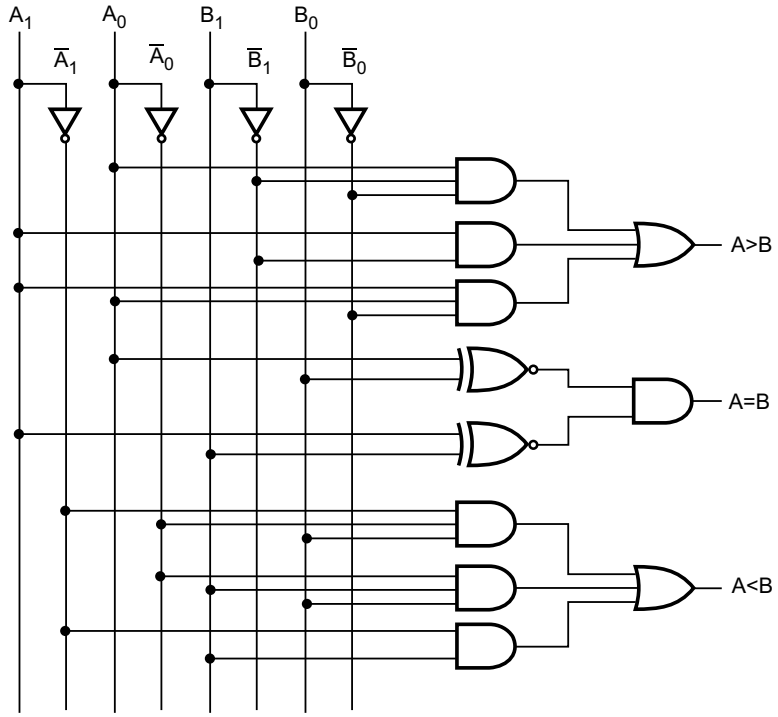
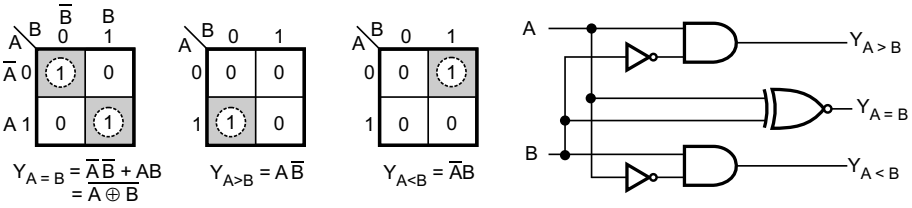


Fig. 4.20.3

Example 4.20.2 *Design a 1-bit comparator using basic gates.* **May-11, Marks 2**

Solution : Consider two one bit number A and B. The truth table is as shown.

Inputs		Outputs		
A	**B**	$Y_{A=B}$	$Y_{A>B}$	$Y_{A<B}$
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0



(a) k-map simplification **(b) Logic diagram**

Fig. 4.20.4

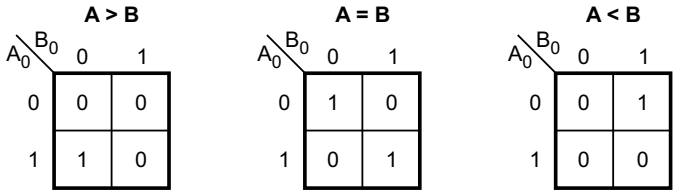
Example 4.20.3 *Design a 5-bit magnitude comparator using comparator IC 7485*

May-08, Marks 8

Solution : Truth Table

A_0	B_0	I(A > B)	I(A = B)	I(A < B)
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

K-map simplification



$$(A > B) = A_0 \overline{B}_0$$

$$(A = B) = \overline{A}_0 \overline{B}_0 + A_0 B_0 = A_0 \odot B_0$$

$$(A > B) = \overline{A}_0 B_0$$

Implementation

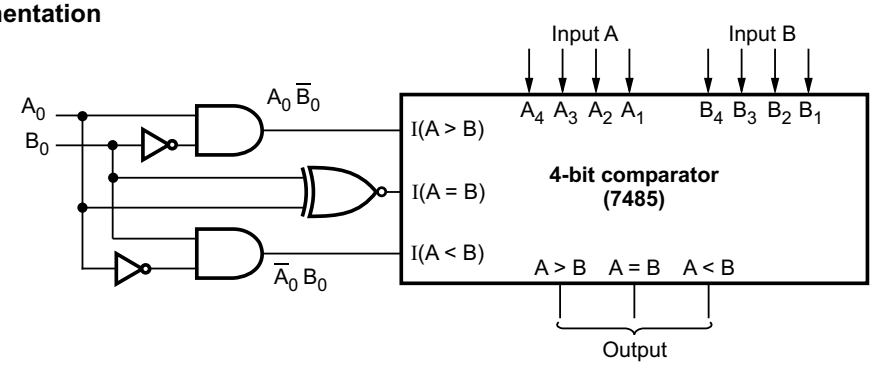


Fig. 4.20.5

Review Questions

1. *What is magnitude comparator ?*
2. *Explain a 2-bit magnitude comparator with gate level circuit and truth table.*
3. *Design a 4-bit word comparator, so that the output follows the table.*

Word	Output
A = B	1 0 0
A > B	0 1 0
A < B	0 0 1

May-12, Marks 16

4.21 Two Marks Questions with Answers

Q.1 *Define a combinational logic circuit. Give an example.* **May-08**

Ans. : When logic gates are connected together to produce a specified output for certain specified combinations of input variables, with no storage involved, the resulting circuit is called 'combinational logic circuit'.

A combinational circuit consists of input variables, logic gates and output variables. For example, consider following Boolean expression,

$$Y = AB + BC + AC$$

The combinational logic circuit for this would require 3 AND gates and 1 OR gate as follows :

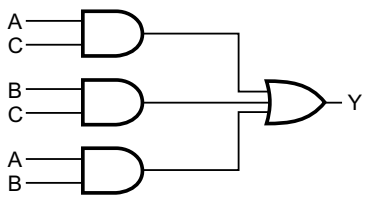


Fig. 4.21.1

Q.2 *Define half adder and full adder.*

Ans. :

1) **Half adder :** The logic circuit which performs the arithmetic sum of two bits is called a half adder.

2) **Full adder :** The logic circuit which performs the arithmetic sum of 3 bits (bit 1 : input 1, bit 2 : input 2, bit 3 : carry from the previous addition) is called a full adder.

Q.3 *Define half subtractor and full subtractor.*

Ans. :

1) **Half subtractor :** It is a combinational circuit that subtracts two bits and produces their difference and borrow.

2) **Full subtractor :** It is a combinational circuit that performs a subtraction between 2 bits. It also takes into account borrow of the lower significant stage.

Q.4 *Give the circuit of a half adder-subtractor.* **May-06**

Ans. : Refer sections 4.3.1 and 4.4.1.

Q.5 *Describe the truth table of a half subtractor and write the Boolean expression corresponding to the difference and the borrow.* **May-05**

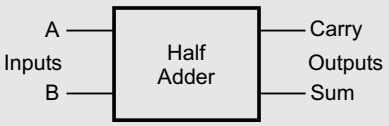
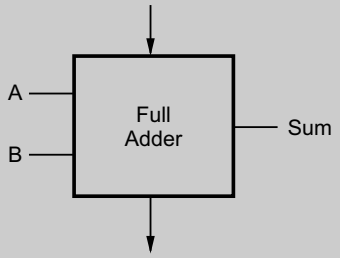
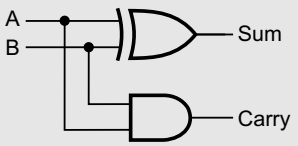
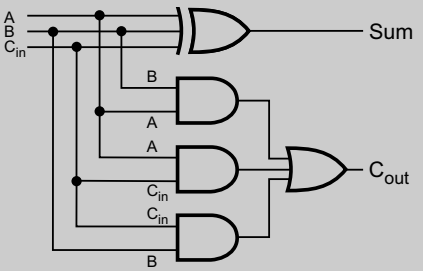
Ans. : Refer section 4.4.1

Q.6 *Implement half adder using gates.* **Dec.-07, May-10**

Ans. : Refer section 4.4.1.

Q.7 *What is the difference between half adder and full adder?* **Dec.-07**

Ans. :

Sr. No.	Half adder	Full adder
1.	Half-adder takes two binary-inputs i.e. augend and addend bits and gives out two binary outputs as sum and carry.	Full-adder, alongwith augend and addend takes third additional bit Cin as input. Cin represents the carry from the previous lower significant position.
2.	Half-adder is not used in practice	Full adder is used in practice
3.	**Block diagram :** 	**Block diagram :** 
4.	**Logic diagram :** 	**Logic diagram :** 

Q.8 *What do you mean by carry propagation delay ?*

Ans. : Definition : In parallel adders, sum and carry outputs of any stage cannot be produced until the input carry occurs. This time delay in the addition process is called **carry propagation delay.** This delay increases with increase in the number of bits to be added in an adder circuit.

Q.9 *What do you mean by comparator ? OR Write a short note on 1-bit comparator.*

Ans. : It is a special combinational circuit designed primarily to compare the relative magnitudes of two binary numbers. An n-bit comparator receives two n-bit numbers, A and B, outputs are : A > B, A = B and A < B. As per the magnitudes of the two numbers, one of the outputs will be high.

Q.10 *What do you mean by encoder ?* **May-12**

Ans. : An encoder is a digital circuit that performs the inverse operation of a decoder. Encoder has 2^n (or fewer) input lines and n output lines. Encoder has enable inputs to activate encoded outputs.

Q.11 *Write a short note on priority encoder. OR What is a priority encoder ?*

Ans. : A priority encoder is an encoder circuit in which if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 4.21.1 Truth table of 4-bit priority encoder

- As per the table, when D_3 is high, regardless of other inputs, the output is 11.
- D_2 has next priority, thus, with $D_2 = 1$ and $D_3 = 0$, output will be 10.
- Output for D_1 is generated only if higher priority inputs are 0, and so on.
- If all inputs are 0, two outputs of the circuit are not used.

Q.12 *What is decoder ?*

Ans. : A decoder is a multiple - input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different. In a binary decoder n-inputs produce 2^n outputs. Usually, a decoder is provided with enable inputs to activate decoded output.

Q.13 *What is binary decoder ?*

Ans. : A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n outputs lines.

Q.14 *What will be the maximum number of outputs for a decoder with a 6 bit data word ?* **May-09**

Ans. : $2^6 = 64$

Q.15 *Mention the uses of decoders.* **Dec.-06**

Ans. : Refer section 4.16.6.

Q.16 *Draw a 4 × 16 decoder constructed with two 3 × 8 decoders.* **May-07**

Ans. : Refer section 4.16.4.

Q.17 *What is a data selector ? or what is multiplexer ? or Why is MUX called as data detector ?* **Dec.-06, May-11**

Ans. :

- Multiplexer is a digital switch. Particularly, it has 2^n input lines and n selection lines whose bit combinations determine which input line is selected and routed onto available only single output line.
- Hence, multiplexer is a selector of one out of several data sources available at its input lines, to connect it to output line. Simply it is a 'many into one' device and also called 'data selector'.

Q.18 *Mention the differences between DMUX and MUX.* **May-07, 09**

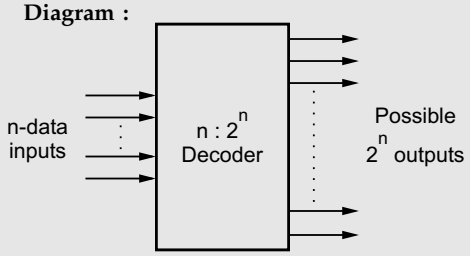
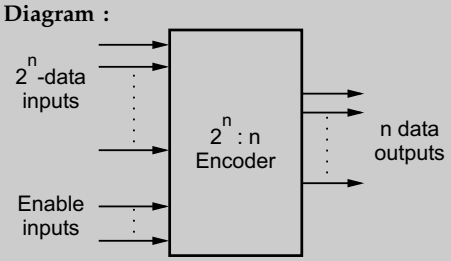
Ans. : Refer section 4.15.

Q.19 *Give an application each for a multiplexer and a demultiplexer.* **Dec.-08, May-12**

Ans. : Refer sections 4.14.7 and 4.15.4.

Q.20 *With a aid of block diagram clearly distinguish between a decoder and encoder.*

Ans. :

Sr. No.	Decoder	Encoder
1.	In decoder one of the output lines is activated corresponding to the binary input.	In encoder, the output lines generate the binary code, corresponding to the input value.
2.	Input of the decoder is an encoded information presented as n inputs producing 2^n possible outputs.	Input of the encoder is a decoded information presented as 2^n inputs producing n possible outputs.
3.	**Diagram :**  **General structure of decoder**	**Diagram :**  **General structure of encoder**
4.	The input code generally has a fewer bits than the output code	The input code generally has more bits than the output code.

Q.21 *Distinguish between a demultiplexer and decoder.* **Dec.-03, 05, 09, May-04,**

Ans. :

Sr. No.	Decoder	Demultiplexer
1.	Decoder is a many inputs to many outputs device.	Demultiplexer is a one input to many outputs device.
2.	There are no selection lines.	The selection of specific output line is controlled by the value of selection lines.

Q.22 *Suggest a solution to overcome the limitation on the speed of an adder.* **Dec.-09**

Ans. : It is possible to increase speed of adder by eliminating inter-stage carry delay. This method utilizes logic gates to look at the lower-order bits of the augend and addend to see if a higher-order carry is to be generated.

❑❑❑

Notes

5 Flip - Flops

Syllabus

Latches, Flip-flops - SR, JK, D, T and Master-Slave - Characteristic table and equation - Application table - Edge triggering - Level triggering - Realization of one flip-flop using other flip-flops.

Contents

5.1 *Introduction*

5.2 *One-Bit Memory Cell*

5.3 *Latches*

5.4 *Flip-Flops* . . . . . ***May-03, 04, 05, 06, 07, 08, 10,*** · · *Marks 8*

. . . . . ***Dec.-03, 05, 06, 08, 09, 10,*** · · · · *Marks 10*

5.5 *Various Representation of Flip-Flops* ***May-04, 05, 06, 07, 08, 09, 10, 11,***

. . . . . ***Dec.-03, 09, 10,*** · · · · · · · · · · · · · *Marks 4*

5.6 *Realization of One Flip-Flop using.* ***May-03, 05, 11, 12,***

Other Flip-Flop . . . . . ***Dec.-03, 04, 08, 09, 11, 12,*** · · · · · *Marks 8*

5.7 *Two Marks Questions with Answers*

5.1 Introduction

There are many applications in which digital outputs are required to be generated in accordance with the sequence in which the input signals are received. This requirement cannot be satisfied using a combinational logic system. These applications require outputs to be generated that are not only dependent on the present input conditions but they also depend upon the past history of these inputs. The past history is provided by feedback from the output back to the input.

Fig. 5.1.1 shows the block diagram of sequential circuit/Finite State Machine (FSM). As shown in the Fig. 5.1.1, memory elements are connected to the combinational circuit as a feedback path.

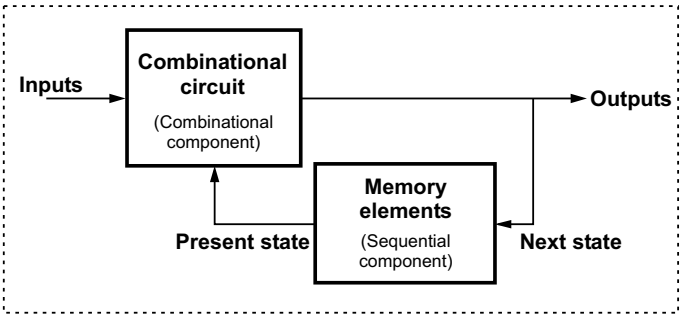


Fig. 5.1.1 Block diagram of sequential circuit / FSM

The information stored in the memory elements at any given time defines the **present state** of the sequential circuit. The present state and the external inputs determine the outputs and the **next state** of the sequential circuit. Thus we can specify the sequential circuit by a time sequence of external inputs, internal states (present states and next states), and outputs. The counters and registers are the common examples of sequential circuits.

The memory element used in sequential circuits is a flip-flop which is capable of storing 1-bit binary information.

5.1.1 Comparison between Combinational and Sequential Logic Circuits

Sr. No.	Combinational circuits	Sequential circuits
1.	In combinational circuits, the output variables are at all times dependent on the combination of input variables.	In sequential circuits, the output variables depend not only on the present input variables but they also depend upon the past history of these input variables.

2.	Memory unit is not required in combinational circuits.	Memory unit is required to store the past history of input variables in the sequential circuit.
3.	Combinational circuits are faster in speed because the delay between input and output is due to propagation delay of gates.	Sequential circuits are slower than the combinational circuits.
4.	Combinational circuits are easy to design.	Sequential circuits are comparatively harder to design.
5.	Parallel adder is a combinational circuit.	Serial adder is a sequential circuit.

Table 5.1.1 Comparison between combinational and sequential circuits

5.1.2 Clock

A clock signal is a particular type of signal that oscillates between a high and a low state and is utilized to co-ordinate actions of circuits. It is produced by a clock generator. The most common clock signal is in the form of a square wave with a 50 % duty cycle, usually with a fixed, constant frequency as shown in Fig. 5.1.2. Circuits using the clock signal for synchronization may become active at either the rising edge, falling edge or in the case of double data rate, both in the rising and in the falling edges of the clock cycle.

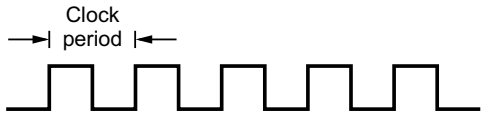


Fig. 5.1.2 Clock signal

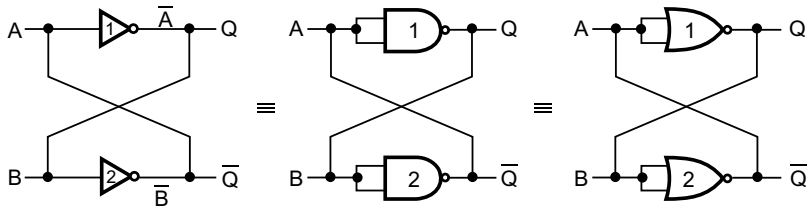
The time required to complete one cycle is called **'clock period'** or **'clock cycle'**. Ideally, the clock signal should have sharp transitions from one level to other as shown in Fig. 5.1.2.

Review Questions

1. *Define sequential logic circuit.*
2. *What is flip-flop ?*
3. *Give the comparison between combinational and sequential logic circuits.*
4. *What is clock ? State its use.*

5.2 One-Bit Memory Cell

The Fig. 5.2.1 shows the basic bistable element used in latches and flip-flops. The basic bistable element has two outputs Q and $\overline{Q}$. It has two cross-coupled inverters, i.e., the output of the first inverter is connected as an input to the second inverter and the output of second inverter is connected as an input to the first inverter.



(a) Using basic inverter **(b) Using NAND gates** **(c) Using NOR gates**

Fig. 5.2.1 Basic bistable element (1-bit memory cell)

The basic bistable element circuit has two stable states logic 0 and logic 1, hence the name 'bistable'. To illustrate this, assume A = 0. When A = 0, the output of inverter 1 is 1 ($\overline{A}$), i.e., Q = 1. Since the output of inverter 1 is the input to the inverter 2, $\overline{A}$ = B = 1. Consequently, the output of inverter 2, i.e., $\overline{B}$ is 0. Since the output of the inverter 2 is connected to the input of the inverter 1, $\overline{Q}$ = $\overline{B}$ = A = 0. We have assumed same value for A. Thus, the circuit is stable with $\overline{Q}$ = A = $\overline{B}$ = 0 and Q = $\overline{A}$ = B = 1. Using similar explanation it is easy to show that if it is assumed that A = 1, the basic bistable element is stable with $\overline{Q}$ = A = $\overline{B}$ = 1 and Q = $\overline{A}$ = B = 0. This is a second stable condition of the basic bistable element.

The two stable states of basic bistable elements are used to store two binary elements, 0 and 1. In positive logic system, state Q = 1 is used to store logic 1, and state Q = 0 is used to store logic 0. It is important to note that the two outputs are complementary. That is when Q = 0, $\overline{Q}$ = 1; and when Q = 1, $\overline{Q}$ = 0.

From the above discussion we can note following things about the basic bistable element.

1. The outputs Q and $\overline{Q}$ are always complementary.
2. The circuit has two stable states. The state corresponds to Q = 1 is referred to as **1 state** or **set state** and state corresponds to Q = 0 is referred to as **0 state** or **Reset state.**
3. If the circuit is in the set (1) state, it will remain in the set state and if the circuit is in the reset (0) state, it will remain in the reset state. This property of the circuit shows that it can store 1-bit of digital information. Therefore, the circuit is called a **1-bit memory cell.**
4. The 1-bit information stored in the circuit is locked or latched in the circuit. Therefore, this circuit is also referred to as a **latch.**

Review Question

1. *Explain the operation of one-bit memory cell.*

5.3 Latches

5.3.1 SR Latch

Fig. 5.3.1 shows SR latch which is 1-bit memory cell. As shown in the Fig. 5.3.1, two inverters 3 and 4 are connected to enter the digital information. Input for gate 3 is **S** and input for gate 4 is **R**. This latch is also called **RS latch.**

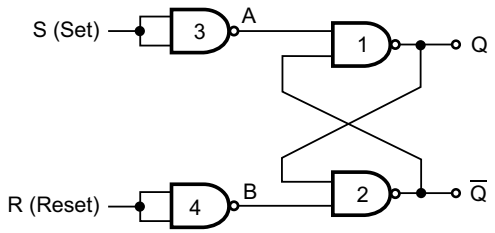


Fig. 5.3.1 SR latch

For understanding the circuit operation, we must first determine the output of NAND gate whose one of the input is logic 0 and accordingly we have to determine the output of other NAND gate in the cross coupled circuit. Because the output of NAND gate is 1 if any one input is 0. The circuit operation is as follows. In Fig. 5.3.2, the output of shaded NAND gate is determined first, and the 0 input that decides the output of shaded NAND as 1 is shown in bold.

Case 1 : S = R = 0

In this case, $\overline{S} = \overline{R} = 1$. If Q is 1, Q and B inputs for NAND gate 2 are both 1 and hence output $\overline{Q} = 0$. Since $\overline{Q} = 0$ and $\overline{S} = 1$, the output of NAND gate 1 is 1, i.e. Q = 1.

If Q is 0, Q and $\overline{R}$ inputs for NAND gate 2 are 0 and 1, and hence output $\overline{Q} = 1$. Since $\overline{Q} = 1$ and $\overline{S} = 1$, the output of NAND gate 1 is 0, i.e., Q = 0.

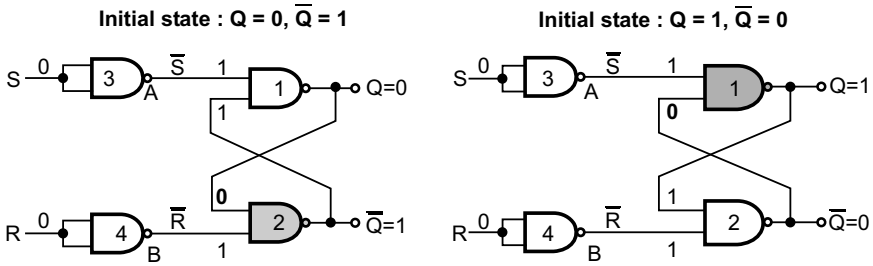


Fig. 5.3.2

This shows that when S = R = 0, the outputs do not change.

Case 2 : S = 1 and R = 0

In this case, $\overline{S} = 0$ and $\overline{R} = 1$. Since $\overline{S} = 0$, the output of NAND gate 1, Q = 1 (Recall that, for NAND any one or more input is 0, the output is

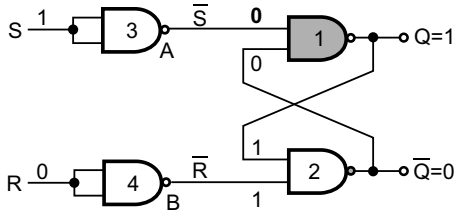


Fig. 5.3.3

1) For NAND gate 2, both inputs Q and $\overline{R}$ are 1, thus output $\overline{Q}$ = 0.

The inputs S = 1 and R = 0, makes Q = 1, i.e., **set** state.

Case 3 : S = 0 and R = 1

In this case, $\overline{S}$ = 1 and $\overline{R}$ = 0. Since $\overline{R}$ = 0, the output of NAND gate 2, $\overline{Q}$ = 1. For NAND gate 1, both inputs $\overline{Q}$ and $\overline{S}$ are 1, thus output Q = 0.

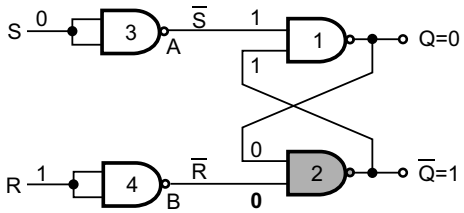


Fig. 5.3.4

The inputs S = 0 and R = 1, makes Q = 0, i.e., **reset** state.

Case 4 : S = 1 and R = 1

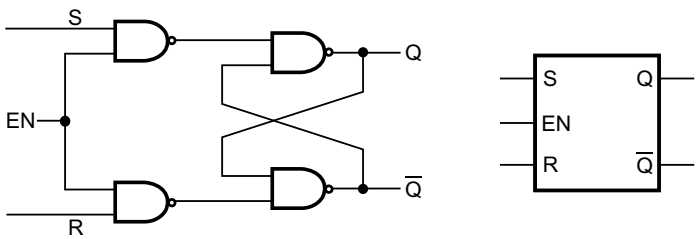
When S = R = 1, both the outputs Q and $\overline{Q}$ try to become 1 which is not allowed and therefore, this input condition is prohibited.

5.3.2 Gated SR Latch

In the SR latch we have seen that output changes occur immediately after the input changes occur i.e. the latch is sensitive to its S and R inputs at all times. However, it can easily be modified to create a latch that is sensitive to these inputs only when an enable input is active. Such a latch with enable input is known as **gated SR latch**. It is as shown in the Fig. 5.3.5. The Table 5.3.1 shows the truth table for gated latch. As shown by truth table, the circuit behaves like a SR latch when EN = 1, and retains its previous state when EN = 0.

EN	S	R	Q_n	Q_{n+1}	State
1	0	0	0	0	No Change (NC)
1	0	0	1	1	
1	0	1	0	0	Reset
1	0	1	1	0	
1	1	0	0	1	Set
1	1	0	1	1	
1	1	1	0	X	Indeterminate
1	1	1	1	X	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	

Table 5.3.1 Truth table for SR latch with enable input



(a) SR latch with enable input using NAND gates **(b) Logic symbol**

Fig. 5.3.5

5.3.3 Gated D Latch

Looking at the truth table of the SR latch we can realize that when both inputs are same the output either does not change or it is invalid (Inputs → 00, no change and inputs → 11, invalid). In many practical applications, these input conditions are not required. These input conditions can be avoided by making them complement of each other. This modified SR latch is known as D latch.

Fig. 5.3.6 shows the D latch. The NAND gates 1, 2, 3 and 4 form the basic SR latch with enable input. The fifth NAND gate is used to provide the complemented inputs.

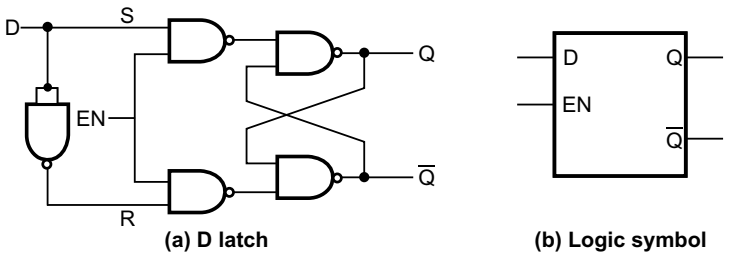


Fig. 5.3.6

As shown in the Fig. 5.3.6, D input goes directly to the S input, and its complement is applied to the R input, through gate 5. Therefore, only two input conditions exists, either S = 0 and R = 1 or S = 1 and R = 0. The truth table for D latch is as shown in the Table 5.3.2.

EN	D	Q_n	Q_{n+1}	State
1	0	X	0	Reset
1	1	X	1	Set
0	X	X	Q_n	No Change (NC)

Table 5.3.2 Truth table for D latch

As shown in the truth table, the Q output follows the D input. For this reason D latch is sometimes called **transparent latch**.

Looking at the truth table for D latch with enable input and simplifying Q_{n+1} function by k-map we get the characteristic equation for D latch with enable input as $Q_{n+1} = EN \cdot D + \overline{EN} \cdot Q_n$. This is illustrated in Fig. 5.3.7.

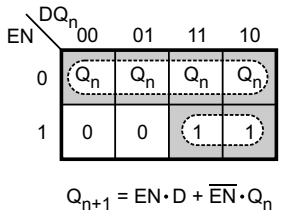


Fig. 5.3.7 Characteristic equation

Review Questions

1. *What is SR latch ? Explain it's operation.*
2. *What is gated SR latch ?*
3. *Explain the working of gated D latch with truth table and characteristic equation.*

5.4 Flip-Flops

May-03, 04, 05, 06, 07, 08, 10, Dec.-03, 05, 06, 08, 09, 10

5.4.1 Latches Vs Flip-Flops

Latches and flip-flops are the basic building blocks of the most sequential circuits. The main difference between latches and flip-flops is in the method used for changing their state.

A simple latch forms the basis for the flip-flop. Latches are controlled by enable signal, and they are level triggered, either positive level triggered or negative level triggered. The output state is free to change according to the S and R input values, when active level is maintained at the enable input. Flip-flops are different from latches. Flip-flops are pulse or clock edge triggered instead of level triggered.

5.4.2 Level and Edge Triggering

Level Triggering

In the level triggering, the output state is allowed to change according to input(s) when active level (either positive or negative) is maintained at the enable input. There are two types of level triggered latches :

- **Positive level triggered :** The output of flip-flop responds to the input changes only when its enable input is 1 (HIGH).

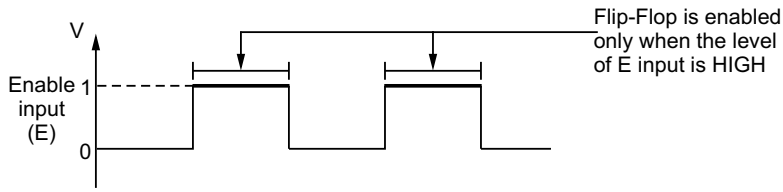


Fig. 5.4.1 Positive level triggering

- **Negative level triggered :** The output of flip-flop responds to the input changes only when its enable input is 0 (LOW).

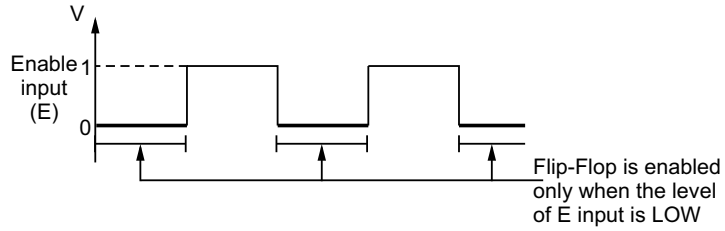


Fig. 5.4.2 Negative level triggering

Edge Triggering

In the edge triggering, the output responds to the changes in the input only at the positive or negative edge of the clock pulse at the clock input. There are two types of edge triggering.

- **Positive edge triggering :** Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.

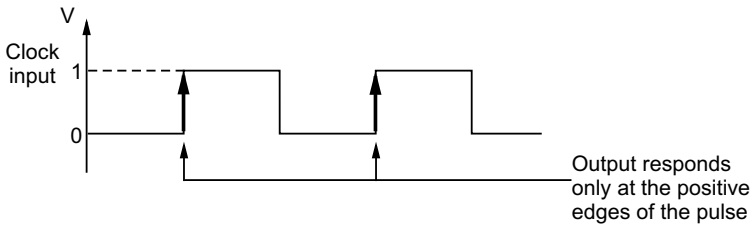


Fig. 5.4.3 Positive edge triggering

- **Negative edge triggering :** Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.

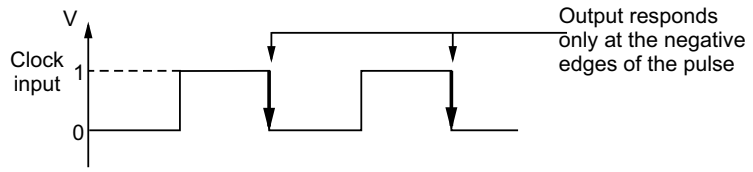


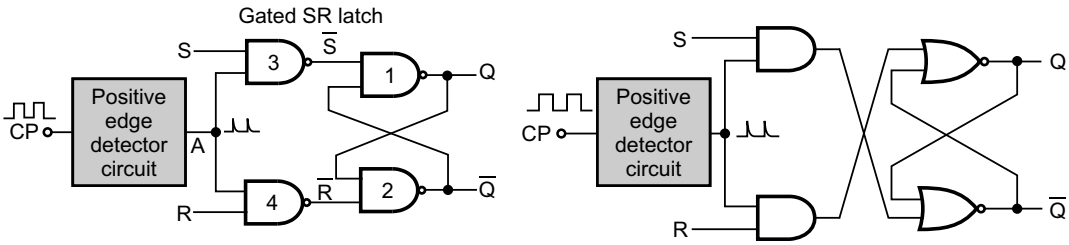
Fig. 5.4.4 Negative edge triggering

5.4.3 SR Flip-Flop

Positive Edge Triggered SR Flip-Flop

The Fig. 5.4.5 shows the positive edge triggered clocked SR flip-flop. The circuit is similar to SR latch except enable signal is replaced by the Clock Pulse (CP) followed by the positive edge detector circuit. The edge detector circuit is a differentiator. The Fig. 5.4.7 shows input and output waveforms for positive edge triggered clocked SR flip-flop. As shown in Fig. 5.4.7 the circuit output responds to the S and R inputs only

at the positive edges of the clock pulse. At any other instants of time, the SR flip-flop will not respond to the changes in input.



(a) SR flip-flop using NAND gates **(b) SR flip-flop using NOR gates**

Fig. 5.4.5 Clocked SR flip-flop

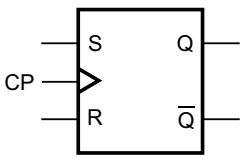
The Fig. 5.4.6 shows the logic symbol and truth table of clocked SR flip-flop.

Case 1 : If S = R = 0 and the clock pulse is applied, the output do not change, i.e. $Q_{n+1} = Q_n$. This is indicated in the first row of the truth table.

Case 2 : If S = 0, R = 1 and the clock pulse is applied, $Q_{n+1} = 0$. This is indicated in the second row of the truth table.

Case 3 : If S = 1, R = 0 and the clock pulse is applied, $Q_{n+1} = 1$. This is indicated in the third row of the truth table.

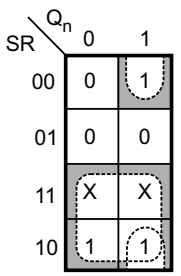
Case 4 : If S = R = 1 and the clock pulse is applied, the state of the flip-flip is undefined and therefore is indicated as indeterminate in the fourth row of the truth table.



(a) Logic symbol

CP	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No Change(NC)
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	X	Indeterminate
↑	1	1	1	X	
0	X	X	0	0	No Change(NC)
0	X	X	1	1	

(b) Truth table for positive edge clocked SR flip-flop



$Q_{n+1} = S + \overline{R}\, Q_n$

(c) Characteristic equation

Fig. 5.4.6

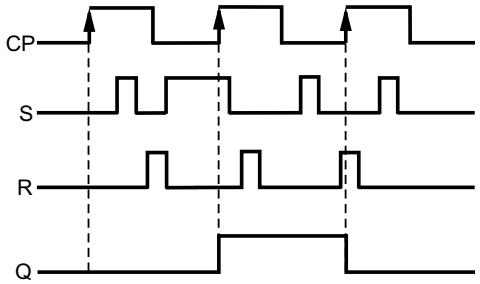
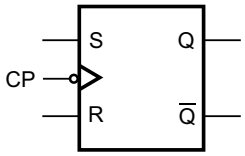


Fig. 5.4.7 Input and output waveforms for positive edge triggered clocked SR flip-flop

Negative Edge Triggered SR Flip-Flop

In the negative edge triggered SR flip-flop, the negative edge detector circuit is used and the circuit output responds at the negative edges of the clock pulse. The Fig. 5.4.8 and Fig. 5.4.9 shows the logic symbol, truth table, and input and output waveforms for negative edge triggered SR flip-flop. The bubble at the clock input indicates that the flip-flop is negative edge triggered.



(a) Logic symbol

CP	S	R	Q_n	Q_{n+1}	State
↓	0	0	0	0	No change(NC)
↓	0	0	1	1	
↓	0	1	0	0	Reset
↓	0	1	1	0	
↓	1	0	0	1	Set
↓	1	0	1	1	
↓	1	1	0	X	Indeterminate
↓	1	1	1	X	
0	X	X	0	0	No change(NC)
0	X	X	1	1	

(b) Truth Table for negative edge clocked SR flip-flop

Fig. 5.4.8

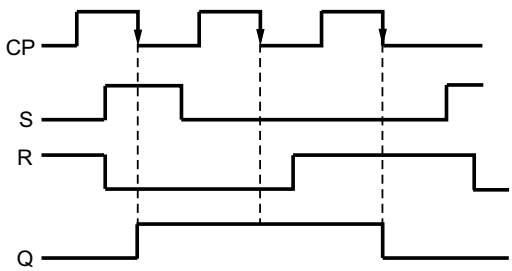


Fig. 5.4.9 Input and output waveforms for negative edge triggered clocked SR flip-flop

Example 5.4.1 *Realize SR flip-flop using NOR gates.* **Dec.-05, Marks 4**

Solution :

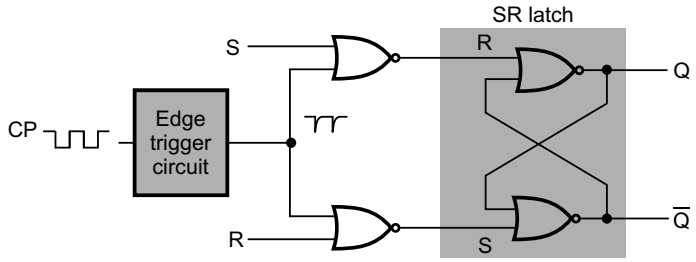
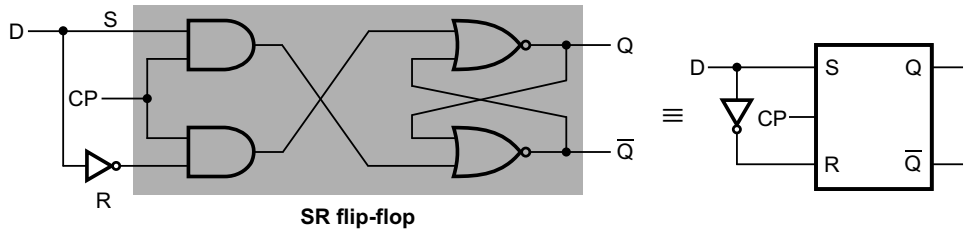


Fig. 5.4.10 Negative edge-triggered SR flip-flop using only NOR gates

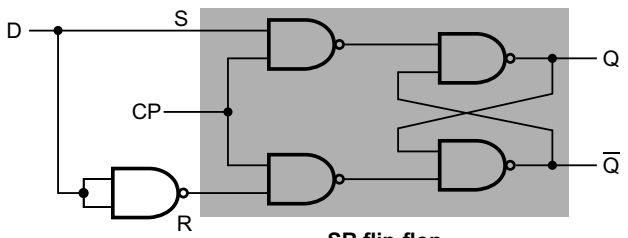
5.4.4 D Flip-Flop

The Fig. 5.4.11 shows the logic diagrams of D flip-flop. The basic building block of D flip-flop is a SR flip-flop. The SR flip-flop has two data inputs S and R. The S input is made high to store 1 in the flip-flop and R input is made high to store 0 in the flip-flop.

Looking at the truth table of the SR flip-flop we can realize that when both inputs are same the output either does not change or it is invalid (Inputs → 00, no change and inputs → 11, invalid). In many practical applications, these input conditions are not required. These input conditions can be avoided by making them complement of each other. This modified SR flip-flop is known as D flip-flop.



(a) D flip-flop



(b) D flip-flop using NAND gates

Fig. 5.4.11

As shown in the Fig. 5.4.11, the D input goes directly to the S input, and its complement is applied to the R input. Due to these connections, only two input conditions exists, either S = 0 and R = 1 or S = 1 and R = 0. The truth table for D flip-flop consider only these two conditions and it is as shown in the Fig. 5.4.12 (b).

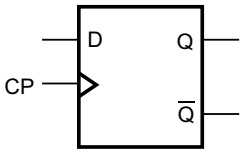


Fig. 5.4.12 (a) Logic symbol

CP	D	Q_{n+1}
↑	0	0
↑	1	1
0	X	Q_n

Fig. 5.4.12 (b) Truth table of D flip-flop

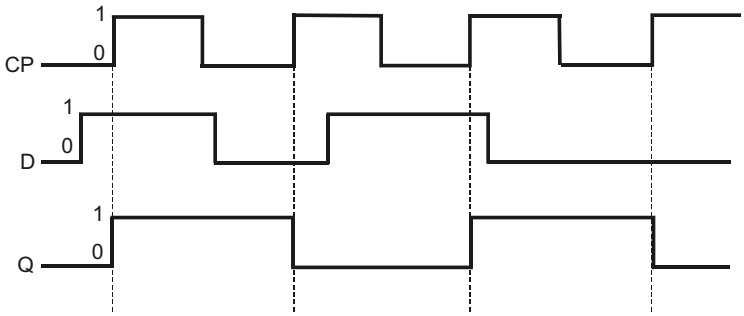


Fig. 5.4.12 (c) Input and output waveforms of clocked D flip-flop

Looking at the truth table for D flip-flop we can realize that Q_{n+1} function follows D input at the positive going edges of the clock pulses. Hence the characteristic equation for D flip-flop is Q_{n+1} = D. However, the output Q_{n+1} is delayed by one clock period. Thus, D flip-flop is also known as **delay flip-flop.**

If we connect the $\overline{Q}$ output of D flip-flop to its D input as shown in the Fig. 5.4.13, the output of D flip-flop will change either from 0 to 1 or from 1 to 0 at every positive edge of the D flip-flop.

Such change in the output is known as **toggling** of the flip-flop output.

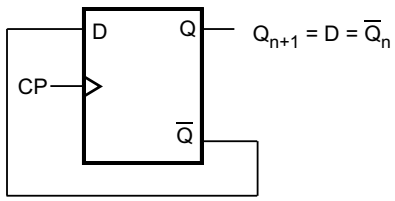
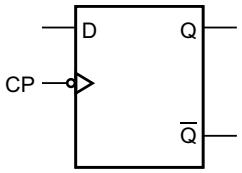


Fig. 5.4.13

Negative Edge Triggered D Flip-Flop

In the previous explanation we have seen the output of D flip-flop is sensitive at the positive edge of the clock input. In case of negative edge triggering, the output is sensitive at the negative edge of the clock



(a) Logic symbol

CP	D	Q_{n+1}
↓	0	0
↓	1	1
0	X	Q_n

(b) Truth table of D flip-flop

Fig. 5.4.14

input. The Fig. 5.4.14 shows the logic symbol and truth table for negative edge triggered D flip-flop and Fig. 5.4.15 shows input and output waveforms for negative edge triggered D flip-flop. The bubble at the clock input indicates that the flip-flop is negative edge triggered.

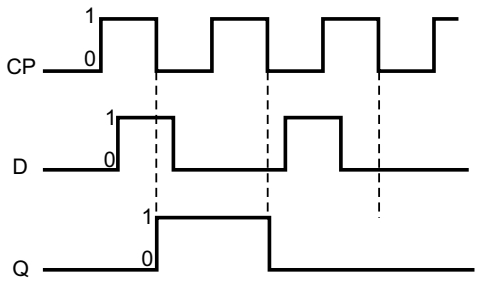


Fig. 5.4.15 Input output waveforms of negative edge triggered D flip-flop

Example 5.4.2 *The D input and a single clock pulse are shown in Fig. 5.4.16. Compare the resulting Q outputs for : Positive edge triggered flip-flop, negative edge triggered flip-flop and pulse triggered master-slave flip-flops. The flip-flops are initially RESET.*

Dec.-04, Marks 3

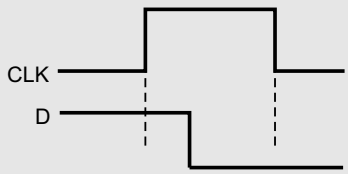


Fig. 5.4.16

Solution :

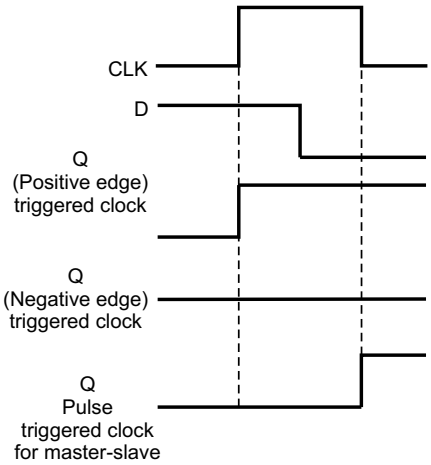


Fig. 5.4.16 (a)

5.4.5 JK Flip-Flop

The uncertainty in the state of an SR flip-flop when S = R = 1 can be eliminated by converting it into a JK flip-flop. The data inputs are J and K which are ANDed with Q

and Q, respectively, to obtain S and R inputs, as shown in the Fig. 5.4.17. Thus, $S = J \cdot \overline{Q}$ and $R = K \cdot Q$.

Let us see the operation of JK flip-flop.

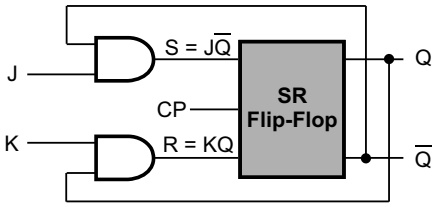


Fig. 5.4.17 JK flip-flop using SR flip-flop

Case 1 : J = K = 0

When J = K = 0, S = R = 0 and according to truth table of SR flip-flop there is no change in the output.

When inputs J = K = 0, output does not change.

Case 2 : J = 1 and K = 0

Q = 0, $\overline{Q}$ = 1 : When J = 1, K = 0 and Q = 0, S = 1 and R = 0. According to truth table of SR flip-flop it is **set** state and the output Q will be 1.

Q = 1, $\overline{Q}$ = 0 : When J = 1, K = 0 and Q = 1, S = 0 and R = 0. Since SR = 00, there is no change in the output and therefore, Q = 1 and $\overline{Q}$ = 0.

The inputs J = 1 and K = 0, makes Q = 1, i.e. **set** state.

Case 3 : J = 0 and K = 1

Q = 0, $\overline{Q}$ = 1 : When J = 0, K = 1 and Q = 0, S = 0 and R = 0. Since SR = 00, there is no change in the output and therefore, Q = 0 and $\overline{Q}$ = 1.

Q = 1, $\overline{Q}$ = 0 : When J = 0, K = 1 and Q = 1, S = 0 and R = 1. According to truth table of SR flip-flop it is a **reset** state and the output Q will be 0.

The inputs J = 0 and K = 1, makes Q = 0, i.e., **reset** state.

Case 4 : J = K = 1

Q = 0, $\overline{Q}$ = 1 : When J = K = 1 and Q = 0, S = 1 and R = 0. According to truth table of SR flip-flop it is a set state and the output Q will be 1.

Q = 1, $\overline{Q}$ = 0 : When J = K = 1 and Q = 1, S = 0 and R = 1. According to truth table of SR flip-flop it is a reset state and the output Q will be 0.

The input J = K = 1, toggles the flip-flop output.

The Fig. 5.4.18 shows the logic symbol, truth table and timing diagram of positive edge triggered JK flip-flop.

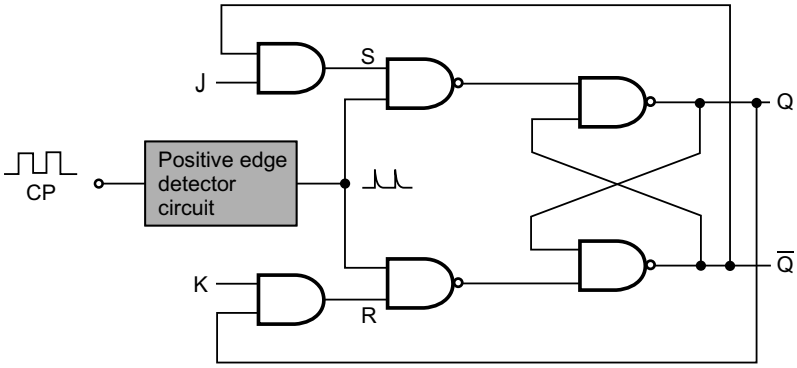


Fig. 5.4.18 (a) Clocked JK flip-flop

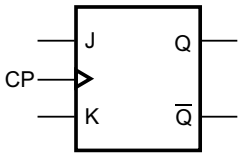


Fig. 5.4.18 (b) Logic symbol

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

≡

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Fig. 5.4.18 (c) Truth table

Q_n \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$Q_{n+1} = \overline{Q}_n J + Q_n \overline{K} = J \overline{Q}_n + \overline{K} Q_n$$

Fig. 5.4.18 (d) Characteristics equation

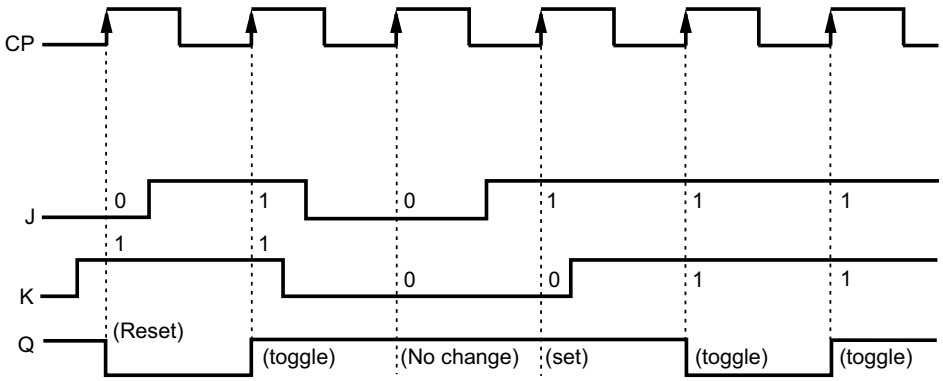


Fig. 5.4.19 Input and output waveforms for positive edge triggered JK flip-flop

Example 5.4.3 *Construct a clocked JK flip-flop which is triggered at the positive edge of the clock pulse from a clocked SR flip-flop consisting of NOR gates.* **May-10, Marks 4**

Solution :

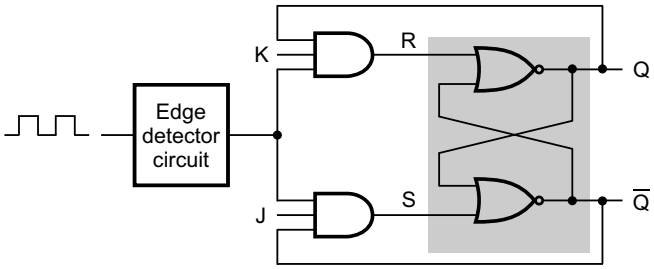


Fig. 5.4.20

5.4.5.1 JK Flip-Flop using NAND Gates

In the previous section we have seen the operation of JK flip-flop using SR flip-flop and AND gates. It is not necessary to use the AND gates of Fig. 5.4.19 (a), since the same function can be performed by adding an extra input terminal to NAND gates 3 and 4 of Fig. 5.4.21. The Fig. 5.4.21 shows the modified circuit of JK flip-flop which has only NAND gates.

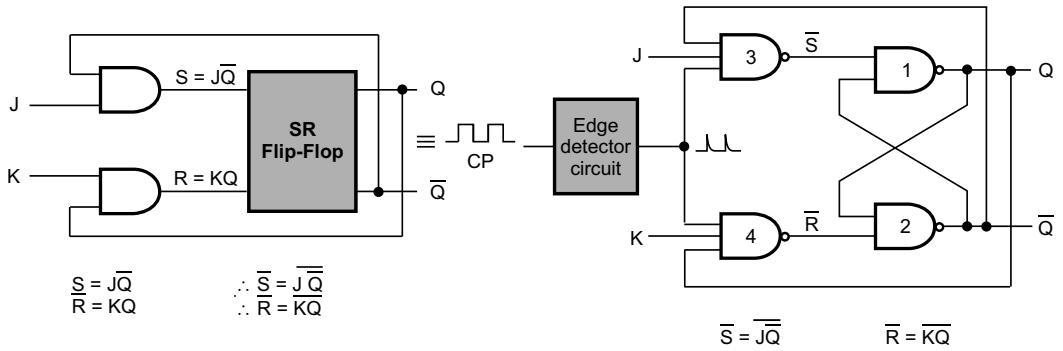


Fig. 5.4.21 JK flip-flop using NAND gates

5.4.5.2 Race-around Condition

In JK flip-flop, when J = K = 1, the output toggles (output changes either from 0 to 1 or from 1 to 0). Consider that initially Q = 0 and J = K = 1. After a time interval Δt equal to the propagation delay through two NAND gates in series, the output will change to Q = 1 and after another time interval of Δt the output will change back to Q = 0. This toggling will continue until the flip-flop is enabled and J = K = 1. At the end of clock pulse the flip-flop is disabled and the value of Q is uncertain. This situation is referred to as the **race-around condition.** This is illustrated in Fig. 5.4.22. This condition exists when $t_p \geq \Delta t$. Thus by keeping $t_p < \Delta t$ we can avoid race around condition.

We can keep $t_p < \Delta t$ by keeping the duration of edge less than Δt. A more practical method for overcoming this difficulty is the use of the Master-Slave (MS) configuration.

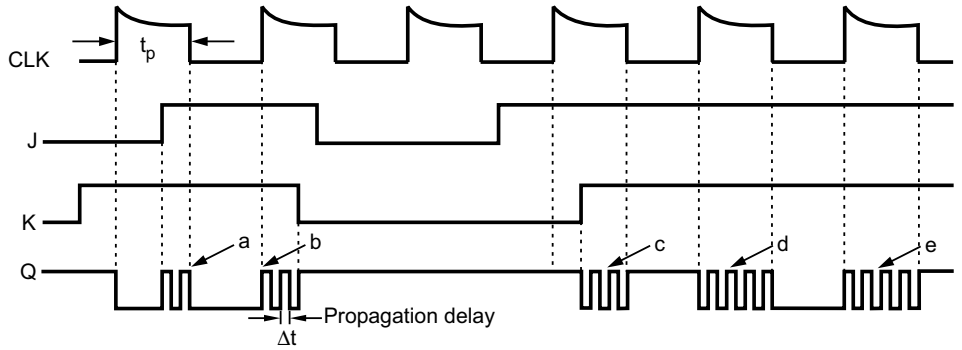


Fig. 5.4.22 Input and output waveforms for clocked JK flip-flop

Example 5.4.4 *Realize a JK flip-flop using only NOR gates.* **Dec.-03, Marks 8**

Solution :

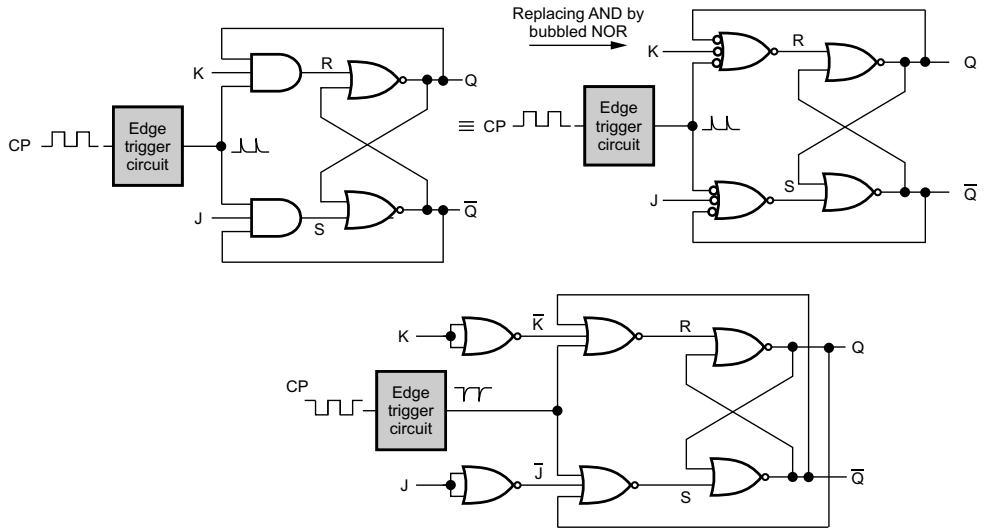


Fig. 5.4.23 Negative edge triggered JK flip-flop using only NOR gates

5.4.6 Master-Slave SR Flip-Flop

A master-slave flip-flop is constructed from two flip-flops. One circuit serves as a master and the other as a slave, and the overall circuit is referred to as a master-slave flip-flop. Fig. 5.4.24 shows SR master-slave flip-flop. It

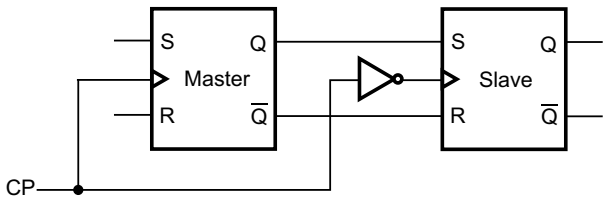


Fig. 5.4.24 Master-slave SR flip-flop

consists of a master flip-flop, a slave flip-flop, and an inverter. Both the flip-flops are positive level triggered, but inverter connected at the clock input of the slave flip-flop forces it to trigger at the negative level.

The output state of the master flip-flop is determined by the S and R inputs at the positive clock pulse. The output state of the master is then transferred as an input to the slave flip-flop. The slave flip-flop uses this input at the negative clock pulse to determine its output state. The Fig. 5.4.25 illustrates the operation of the master-slave flip-flop.

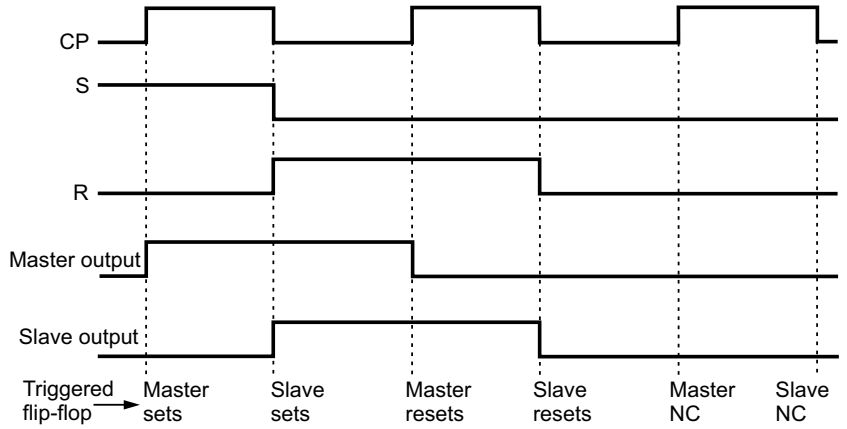


Fig. 5.4.25 Input and output waveforms for master-slave flip-flop

5.4.7 Master-Slave JK Flip-Flop

Fig. 5.4.26 shows one way to build a JK master-slave flip-flop. It consists of clocked JK flip-flop as a master and clocked JK flip-flop as a slave. The output of the master flip-flop is fed as an input to the slave flip-flop. As shown in the Fig. 5.4.26, clock signal is connected directly to the master flip-flop, but it is connected through inverter to the slave flip-flop. Therefore, the information present at the J and K inputs is transmitted to the output of master flip-flop on the positive clock pulse and it is held there until the negative clock pulse occurs, after which it is allowed to pass through to the output of slave flip-flop.

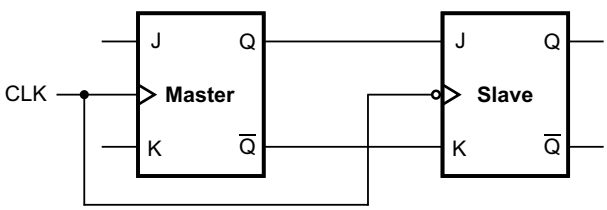


Fig. 5.4.26 Master-slave JK flip-flop

When J = 1 and K = 0, the master sets on the positive clock. The high Y output of the master drives the J input of the slave, so at negative clock, slave sets, copying the action of the master.

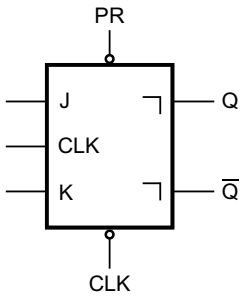
When J = 0 and K = 1, the master resets on the positive clock. The high $\overline{Y}$ output of the master goes to the K input of the slave. Therefore, at the negative clock slave resets, again copying the action of the master.

When J = 1 and K = 1, master toggles on the positive clock and slave then copies the output of master on the negative clock.

When J = K = 0, the output of master remains same at the positive clock. Thus the output of slave also remains same at the negative clock. The Fig. 5.4.27 shows the truth table and symbol for master-slave JK flip-flop.

CLK	J	K	Q_{n+1}
⎍	0	0	Q_n (No change)
⎍	0	1	0 (Reset)
⎍	1	0	1 (Set)
⎍	1	1	$\overline{Q}_n$ (Toggle)

(a) Truth table



(b) Symbol

Fig. 5.4.27

Note : It is important to note that clock input of the master-slave flip-flop is not edge triggered.

5.4.8 T Flip-Flop

T flip-flop is also known as '**Toggle flip-flop**'. The T flip-flop is a modification of the JK flip-flop. As shown in the Fig. 5.4.28, the T flip-flop is obtained from a JK flip-flop by connecting both inputs, J and K together.

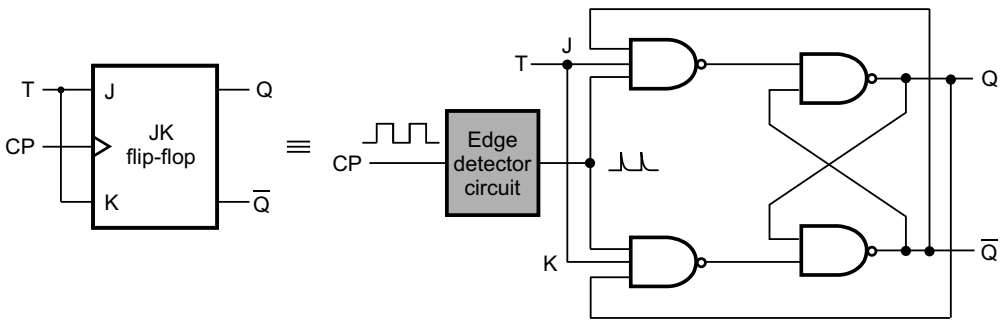
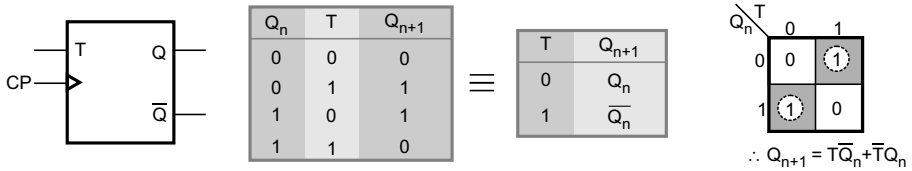


Fig. 5.4.28 T flip-flop using NAND gates

When T = 0, J = K = 0 and hence there is no change in the output. When T = 1, J = K = 1 and hence output toggles.

The Fig. 5.4.29 shows logic symbol, truth table and the characteristic equation for T flip-flop.



(a) Logic symbol b) Truth table (c) Characteristic equation

Fig. 5.4.29

Example 5.4.5 *Refer Fig. 5.4.30 and determine the Q output waveform if the flip-flop starts out RESET.* **Dec.-04, Marks 4**

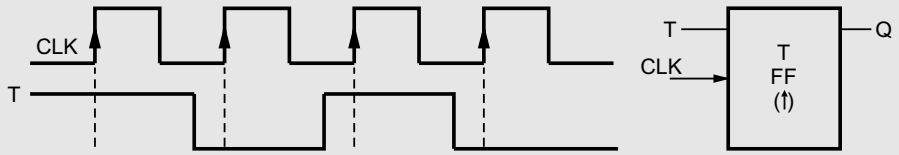


Fig. 5.4.30

Solution :

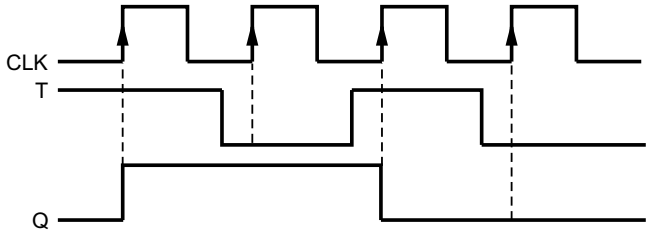
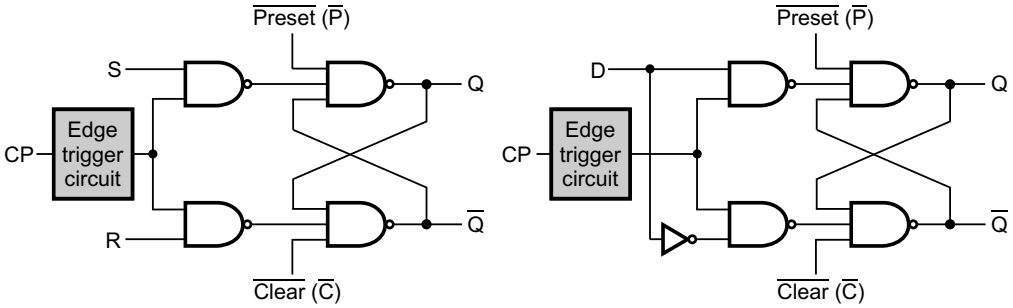


Fig. 5.4.30 (a)

5.4.9 Asynchronous or Direct Inputs

For the flip-flops discussed so far, the SR, D, JK, and T, the inputs are called synchronous inputs because data on these inputs are transferred to the flip-flop's output only on the triggering edge of the clock pulse; that is, the data are transferred synchronously with the clock.

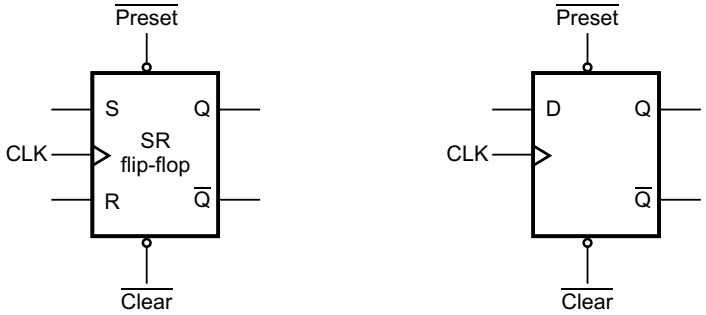
When power is turn **ON**, the state of the flip-flop is uncertain. It may come to set (Q = 1) or reset (Q = 0) state. In many applications, it is necessary to initially set or reset the flip-flop. Such initial state of flip-flop can be accomplished by using the direct or asynchronous inputs of the flip-flop. These inputs are : $\overline{\text{preset}}$ ($\overline{P}$) and $\overline{\text{clear}}$ ($\overline{C}$). They can be applied at any time between clock pulses and are not in synchronism with the clock.



(a) SR flip-flop (b) D flip-flop

Fig. 5.4.31

The Fig. 5.4.31 shows the SR and D flip-flops with preset and clear inputs. These are active-low inputs and thus when $\overline{P} = \overline{C} = 1$, the circuit operates in accordance with the truth table of SR flip-flop. If $\overline{P} = 1$ and $\overline{C} = 0$, the flip-flop is reset and $\overline{P} = 0$ and $\overline{C} = 1$, the flip-flop is set.



(a) Logic symbol for SR flip-flop (b) Logic symbol for D flip-flop

Fig. 5.4.32 Logic symbol

Note : Condition P = C = 0 must not be used, since this leads to an uncertain state.

Review Questions

1. *Explain the working of a master-slave JK flip-flop.* **May-03, 05, 08, Dec.-03, 06, 08, 09, Marks 6**
2. *Realize a SR flip-flop using NAND gates and explain its operation.* **Dec.-03, Marks 10**
3. *Derive the characteristic equation of a SR flip-flop.* **May-04, Marks 8**
4. *With reference to a JK flip-flop, what is racing ?* **May-04, Dec.-08, Marks 2**
5. *Differentiate between flip-flop and latch.* **May-05, Marks 2**
6. *Describe the input and output action of JK master/slave flip-flops.* **May-05, Marks 8**

7. *Draw the circuit for SR flip-flop.* **May-05, 10, Marks 2**
8. *Realize SR flip-flop using NOR gates and explain its operation.* **Dec.-05, Marks 8**
9. *Explain the operation of JK master slave flip-flop with suitable diagrams.* **Dec.-05, Marks 8**
10. *What is the drawback of SR flip-flop ? How is this minimized ?* **May-06, Marks 2**
11. *Explain the working of JK flip-flop.* **May-06, Marks 8**
12. *What is race around condition and how is it overcome ? Explain these concepts with relevant timing diagrams.* **May-06, 08, Dec.-10, Marks 8**
13. *Draw the clocked RS flip-flop and explain with truth table.* **May-07, Marks 8**
14. *How does a JK flip-flop differ from the SR flip-flop in its basic operation ?* **May-07, Marks 2**
15. *Draw the logic symbol and truth table of a D flip-flop.* **May-08, Marks 2**
16. *Draw the logic circuit of a clocked JK flip-flop.* **May-08, Marks 2**
17. *Draw the logic diagram of a D-FF using NAND gates and explain.* **Dec.-08, Marks 6**
18. *Mention any two differences between the edge triggering and level triggering.* **May-10, Marks 2**
19. *Write down the characteristic equation and truth table for the JK flip-flop.* **May-10, Marks 4**
20. *Mention the major application of master slave flip-flop.* **Dec.-10, Marks 2**
21. *Explain the different types of triggering with waveforms.*
22. *Draw and explain the working of clocked D flip-flop.*
23. *How can race condition be avoided in flip-flop ?*
24. *Explain the T flip-flop.*
25. *Explain the operation of master-slave SR flip-flop.*
26. *Explain the use of asynchronous inputs.*

5.5 Various Representation of Flip-Flops

May-04, 05, 06, 07, 08, 09, 10, 11, Dec.-03, 09, 10

5.5.1 Characteristic Equations of Flip-Flops

The characteristic equations for various flip-flops are summarized in Table 5.5.1.

Flip-Flop	Characteristic equation
SR	$Q_{n+1} = S + \overline{R} Q_n$
D	$Q_{n+1} = D$
JK	$Q_{n+1} = J \overline{Q}_n + \overline{K} Q_n$
T	$Q_{n+1} = T \overline{Q}_n + \overline{T} Q_n$

Table 5.5.1 Characteristic equation of flip-flops

5.5.2 Flip-Flops as Finite State Machines

In a sequential logic circuit Finite State Machine (FSM) the value of all the memory elements at a given time define the state of that circuit at that time. In FSM, the functional behaviour of the circuit is represented using the state transition diagram. Let us the representation of flip-flops in their state transition diagrams.

The Fig. 5.5.1 shows the state transition diagram for all flip-flops. Here, the 0 and the 1 in the circle represents the two states of the flip-flops and the arcs with arrow heads indicate the state transitions for specific inputs of the flip-flop. For example, when SR flip-flop is in state 0, it goes to state 1 if SR inputs are 10, i.e. S = 1 and R = 0.

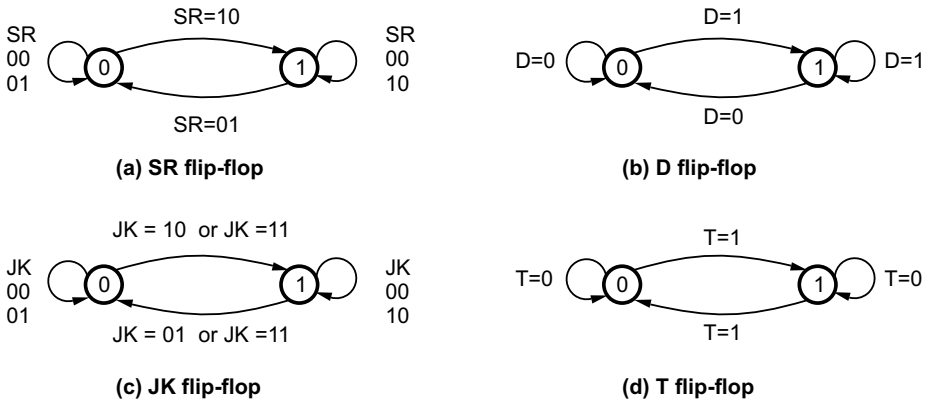


Fig. 5.5.1 State transition diagrams of flip-flops

5.5.3 Flip-Flop Excitation Table

During the design process we know, from the transition table, the sequence of states, i.e., the transition from each present state to its corresponding next state. From this information we wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a table is known as an excitation table of the flip-flop.

We can derive the excitation tables for flip-flops from their truth tables. The excitation table consists of two columns Q_n and Q_{n+1}, and a column for each input to show how the required transition can be achieved. Let us see the truth tables and excitation tables for RS, JK, D and T flip-flops.

5.5.3.1 SR Flip-Flop

Table 5.5.2 (a) and (b) show the truth table and excitation tables for SR flip-flop, respectively. As shown in the table, there are four possible transitions from the present state to the next state. For each transition, the required input condition is derived from the information available in the truth table. Let us see the process by examining each case.

S	R	Q_{n+1}
0	0	Q_n
1	0	1
0	1	0
1	1	*

(a) SR truth table

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

(b) SR excitation table

Table 5.5.2

Note : The symbol "X" in the table represents a don't care condition, i.e., it indicates that to get required output it does not matter whether the input is either 1 or 0.

0 → 0 Transition : The present state of the flip-flop is 0 and is to remain 0 when a clock pulse is applied. Looking at truth table of SR flip-flop we can understand that, this can happen either when R = S = 0 (no-change condition) or when R = 1 and S = 0. Thus, S has to be at 0, but R can be at either level. The table indicates this with a "0" under S and an "X" (don't care) under R.

0 → 1 Transition : The present state is 0 and is to change to 1. This can happen only when S = 1 and R = 0 (set condition). Therefore, S has to be 1 and R has to be 0 for this transition to occur.

1→ 0 Transition : The present state is 1 and is to change to a 0. This can happen only when S = 0 and R = 1 (reset condition). Therefore, S has to be 0 and R has to be 1 for this transition to occur.

1 → 1 Transition : The present state is 1 and is to remain 1. This can happen either when S = 1 and R = 0 (set condition) or when S = 0 and R = 0 (no change condition). Thus R has to be 0, but S can be at either level. The table indicates this with a "X" under S and "0" under R.

5.5.3.2 JK Flip-Flop

The truth table and excitation table for JK flip-flop are shown in Table 5.5.3 (a) and (b) respectively. Let us examine each case.

0 → 0 Transition : When both present state and next state are 0, the J input must remain at 0 and the K input can be either 0 and 1.

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

(a) JK truth table

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

(b) JK excitation table

Table 5.5.3

0 → 1 Transition : The present state is 0 and is to change to 1. This can happen either when J = 1 and K = 0 (set condition) or when J = K = 1 (toggle condition). Thus, J has to be 1, but K can be at either level for this transition to occur.

1→ 0 Transition : The present state is 1 and is to change to 0. This can happen either when J = 0 and K = 1 or when J = K = 1. Thus, K has to be 1 but J can be at either level.

1→ 1 Transition : When both present state and next are 1, the K input must remain at 0 while the J input can be 0 or 1.

As seen from Table 5.5.3, the excitation table for JK flip-flop has more don't care conditions than the excitation table for RS flip-flop. The don't care terms usually simplify the function. Therefore, the combinational circuits using JK flip-flops for the input functions are likely to be simpler than those using RS flip-flops.

5.5.3.3 D Flip-Flop

The Table 5.5.4 (a) and (b) show the truth table and excitation table for D flip-flop, respectively. In D flip-flop, the next state is always equal to the D input and it is independent of the present state. Therefore, D must be 0 if Q_{n+1} has to be 0, and 1 if Q_{n+1} has to be 1, regardless of the value of Q_n.

D	Q_{n+1}
0	0
1	1

(a) D truth table

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

(b) D excitation table

Table 5.5.4

5.5.3.4 T Flip-Flop

The Table 5.5.5 (a) and (b) show the truth table and the excitation table for T flip-flop, respectively. We know that when input T = 1, the state of the flip-flop is complemented; when T = 0, the state of the flip-flop remains unchanged. Therefore, for 0 → 0 and 1 → 1 transitions T must be 0 and for 0 →1 and 1→ 0 transitions T must be 1.

T	Q_{n+1}
0	Q_n
1	$\overline{Q}_n$

(a) T truth table

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

(b) T excitation table

Table 5.5.5

Example 5.5.1 *Show that the characteristic equation of $Q'_{(t+1)}$ of JK flip-flop is* $Q'_{(t+1)} = J'Q' + KQ$ **Dec.-07, Marks 4**

Solution : From Fig. 5.5.2 we can write truth table for JK flip flop as shown below.

Q	J	K	$Q_{(t+1)}$	$\overline{Q}_{(t+1)}$
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

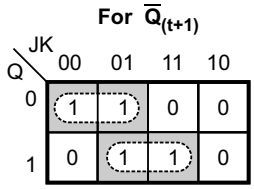


Fig. 5.5.2

$$\therefore \quad \overline{Q}_{(t+1)} = \overline{J}\,\overline{Q} + KQ$$

Review Questions

1. *Give the excitation table of SR flip-flop.* **Dec.-03, Marks 2**
2. *Give the characteristic equation and state diagram of JK flip-flop.* **May-04, 10, 11, Dec.-09, Marks 2**
3. *Give the excitation table for JK flip-flop.* **May-05, 06, 09, Dec.-10, Marks 2**
4. *Obtain the excitation table of D flip-flop.* **May-06, Dec.-10, Mark 1**
5. *Give the state diagram of JK flip-flop.* **May-07, Marks 2**
6. *What is excitation table ?* **May-08, Marks 2**
7. *Give state diagram of JK flip-flop.* **May-10, Marks 2**
8. *Draw state diagram of SR flip-flop.* **Dec.-10, Marks 2**
9. *Draw the truth table and excitation table of T flip-flop.* **Dec.-10, Marks 2**
10. *State the characteristics equations of various flip-flops.*
11. *Give the state diagram of various flip-flops.*
12. *Obtain the excitation table for T flip-flop.*

5.6 Realization of One Flip-Flop using Other Flip-Flop

May-03, 05, 11, 12, Dec.-03, 04, 08, 09, 11, 12

It is possible to convert one flip-flop into another flip-flop with some additional gates or simply doing some extra connection. Let us see few conversions among flip-flops.

5.6.1 SR Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in Table 5.6.1.

Input	Present state	Next state	Flip-flop inputs	
D	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

Table 5.6.1

K-map simplification

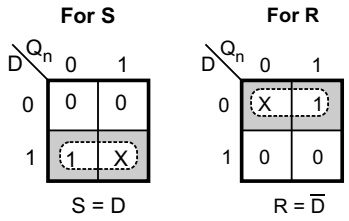


Fig. 5.6.1

Logic diagram

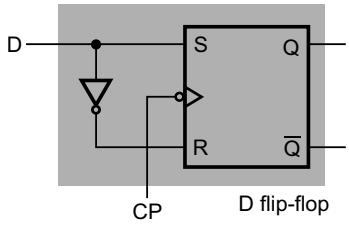


Fig. 5.6.2 SR to D flip-flop conversion

5.6.2 SR Flip-Flop to JK Flip-Flop

The excitation table for above conversion is as shown in Table 5.6.2.

Inputs		Present state	Next state	Flip-flop inputs	
J	K	Q_n	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Table 5.6.2

K-map simplification

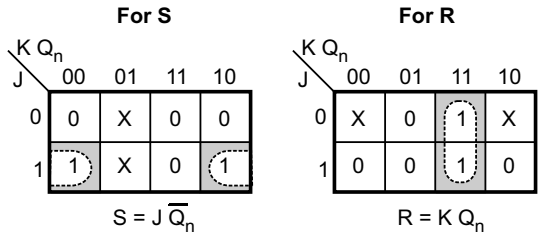


Fig. 5.6.3

Logic diagram

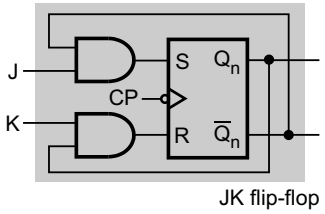


Fig. 5.6.4 SR to JK flip-flip conversion

5.6.3 SR Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in the Table 5.6.3.

Input	Present state	Next state	Flip-flop inputs	
T	Q_n	Q_{n+1}	S	R
0	0	0	0	X
0	1	1	X	0
1	0	1	1	0
1	1	0	0	1

Table 5.6.3

K-map simplification

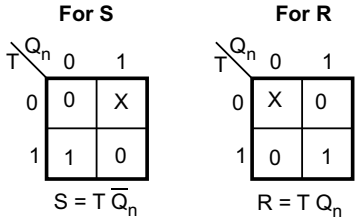


Fig. 5.6.5

Logic diagram

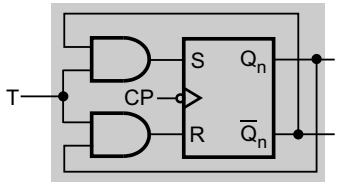


Fig. 5.6.6 SR to T flip-flop conversion

If we apply clock pulses to the circuit, the circuit output will toggle from 0 to 1 and 1 to 0. Thus, we can build 1-bit counter using SR flip-flop by converting it to T flip-flop.

Example 5.6.1 *Prepare the truth table for the circuit of Fig. 5.6.7 and show that it acts as a T-type flip-flop.*

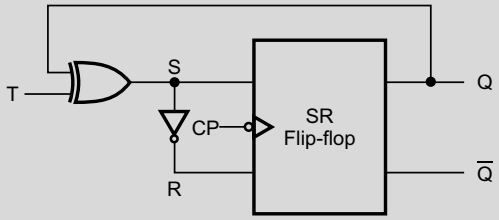


Fig. 5.6.7

Solution : For SR flip-flop,

$$Q_{n+1} = S + \overline{R}\, Q_n \quad \text{... Characteristics equation}$$

$$= S + S\, Q_n \quad \because R = \overline{S}$$

$$= S(1 + Q_n) = S$$

We have, $S = Q_n \oplus T$

$$\therefore \quad Q_{n+1} = Q_n \oplus T$$

$$= T\overline{Q}_n + \overline{T}\, Q_n \quad \text{... Characteristic equation of T flip-flop.}$$

C_p	T	Q_n	$S = Q_n \oplus T$	$R = \overline{S}$	$Q_{n+1} = S$
↓	0	0	0	1	0
↓	0	1	1	0	1
↓	1	0	1	0	1
↓	1	1	0	1	0

Table 5.6.4 Truth table for the given circuit

Looking at column 1 and column 5 of the Table 4.13 we can conclude that when T = 0, the output does not change and when T = 1, the output toggles. Thus, the given circuit acts as a T flip-flop. This is another way of implementing T flip-flop using SR flip-flop.

5.6.4 JK Flip-Flop to T Flip-Flop

May-12

The excitation table for above conversion is as shown in Table 5.6.5.

Input	Present state	Next state	Flip-flop inputs	
T	Q_n	Q_{n+1}	J_A	K_A
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

Table 5.6.5

K-map simplification

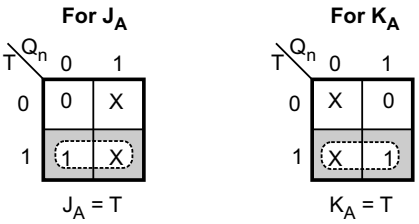


Fig. 5.6.8

Logic diagram

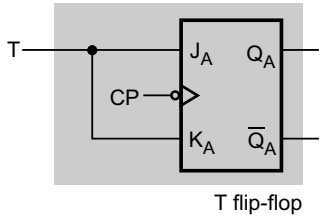


Fig. 5.6.9 JK to T flip-flop conversion

5.6.5 JK Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in the Table 5.6.6.

Input	Present state	Next state	Flip-flop inputs	
D	Q_n	Q_{n+1}	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

Table 5.6.6

K-map simplification

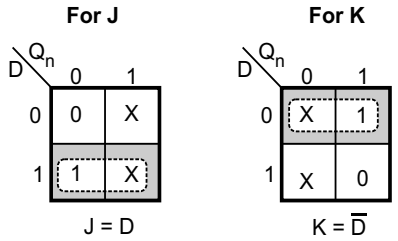


Fig. 5.6.10

Logic diagram

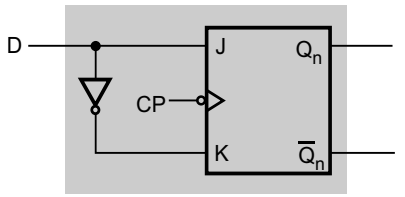


Fig. 5.6.11 JK to D flip-flop conversion

5.6.6 D Flip-Flop to T Flip-Flop

The excitation table for above conversion is as shown in the Table 5.6.7.

Input	Present state	Next state	Flip-flop input
T	Q_n	Q_{n+1}	D
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 5.6.7

K-map simplification

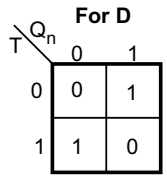


Fig. 5.6.12

Logic diagram

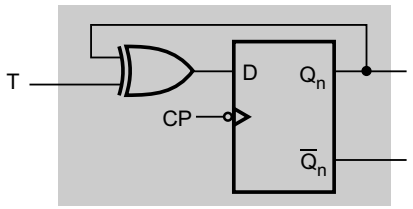


Fig. 5.6.13 D to T flip-flop conversion

$$D = \overline{T}Q_n + T\overline{Q}_n = T \oplus Q_n$$

Example 5.6.2 *Analyze the circuit and prove that it is equivalent to T flip-flop.*

Solution : To analyze the circuit means to derive the truth table for it.

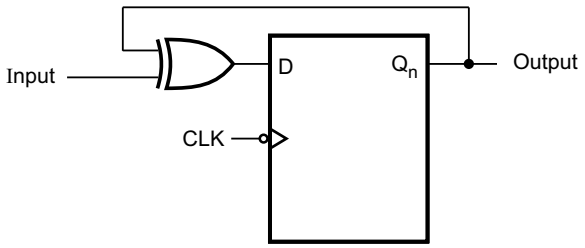


Fig. 5.6.14

We have, $D = \text{Input} \oplus Q_n$

CLK	Input	Q_n	$D = \text{Input} \oplus Q_n$	Q_{n+1}	
↓	0	0	0	0	When input is 0 output does not change
↓	0	1	1	1	
↓	1	0	1	1	When input is 1 output toggles
↓	1	1	0	0	

Table 5.6.8 Truth table for given circuit

In the above circuit, output does not change when input is 0 and it toggles when input is 1. This is the characteristics of T flip-flop. Hence, the given circuit is T flip-flop constructed using D flip-flop.

5.6.7 T Flip-Flop to D Flip-Flop

The excitation table for above conversion is as shown in the Table 5.6.9.

Input	Present state	Next state	Flip-flop input
D	Q_n	Q_{n+1}	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Table 5.6.9

K-map simplification

For T

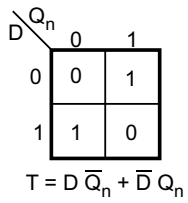


Fig. 5.6.15

Logic diagram

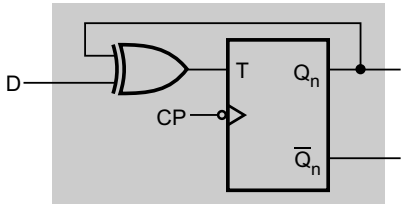


Fig. 5.6.16 T to D flip-flop conversion

5.6.8 JK Flip-Flop to SR Flip-Flop

The excitation table for above conversion is as shown in Table 5.6.10.

Inputs		Present state	Next state	Flip-flop inputs	
S	R	Q_n	Q_{n+1}	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	0	X	X	X
1	1	1	X	X	X

Table 5.6.10 Excitation table for JK to SR conversion

K-map simplification

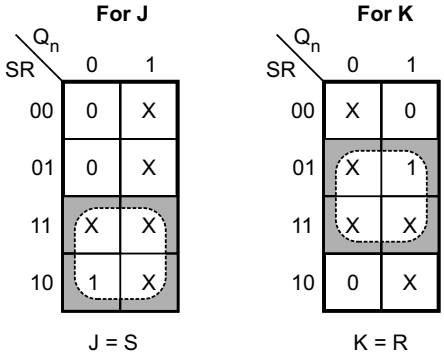


Fig. 5.6.17

Logic diagram

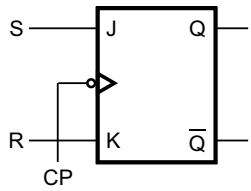


Fig. 5.6.18 JK to SR

5.6.9 D Flip-Flop to SR Flip-Flop

The excitation table for above conversion is as shown in the Table 5.6.11.

Inputs		Present state	Next state	Flip-flop input
S	**R**	Q_n	Q_{n+1}	**D**
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	X	X
1	1	1	X	X

Table 5.6.11 Excitation table for D to SR conversion

K-map simplification

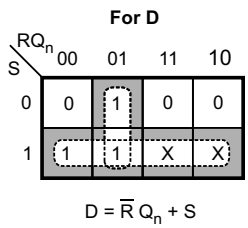


Fig. 5.6.19

Logic diagram

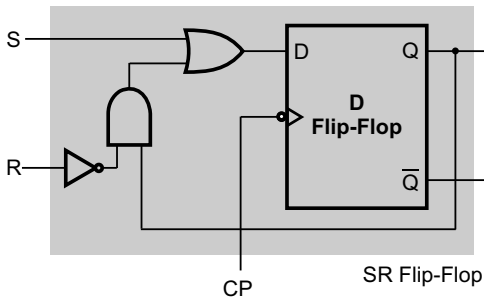


Fig. 5.6.20 D to SR flip-flop conversion

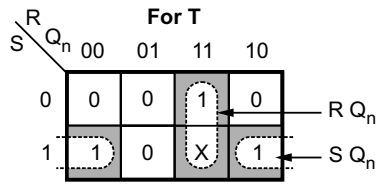
5.6.10 T Flip-Flop to SR Flip-Flop

The excitation table for conversion of T FF into an SR FF is as shown in the table.

Inputs		Present state	Next state	Flip-flop input
S	R	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	X	x
1	1	1	X	x

Table 5.6.12 Excitation table for T to SR conversion

K-map simplification



$$\therefore \quad T = RQ_n + S\,\overline{Q}_n$$

Fig. 5.6.21 (a)

Logic diagram

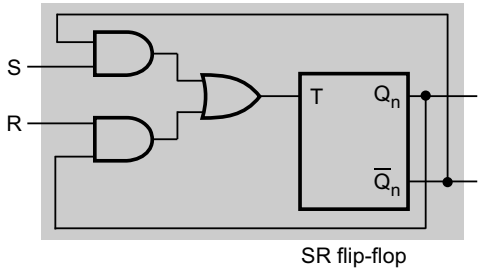


Fig. 5.6.21 (b) SR flip-flop

5.6.11 D Flip-Flop to JK Flip-Flop

The excitation table for conversion of D flip-flop to JK flip-flop :

Inputs		Present state	Next state	Flip-flop input
J	K	Q_n	Q_{n+1}	D
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map simplification **Logic diagram**

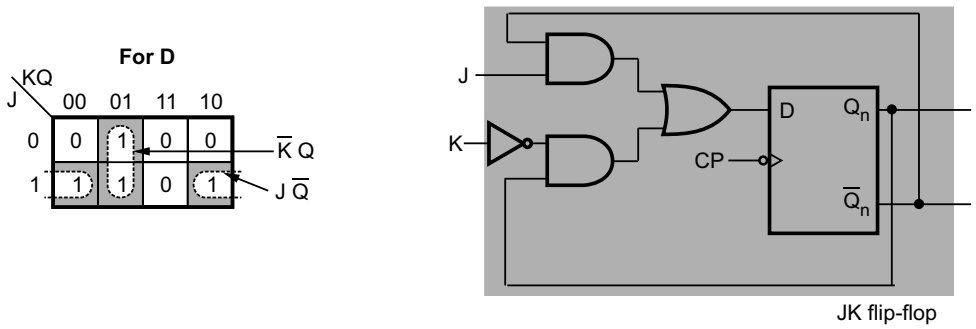


Fig. 5.6.22

Review Questions

1. *Convert JK flip-flop to D flip-flop.* **May-03, 05, Dec.-03, Marks 2**
2. *Convert D flip-flop into T flip-flop.* **Dec.-03, 12, May-11, Marks 6**
3. *Convert JK flip-flop to T flip-flop.* **Dec.-04, 10, May-12, Marks 2**
4. *Convert a SR flip-flop into a D flip-flop.* **Dec.-08, Marks 8**
5. *Convert a T-FF into an S-R FF. Draw the circuit.* **Dec.-08, Marks 2**
6. *How will you convert a D flip-flop into JK flip-flop ?* **Dec.-09, 11, Marks 8**
7. *Convert SR flip-flop into JK flip-flop.*
8. *Convert SR flip-flop into T flip-flop.*
9. *Convert T flip-flop into D flip-flop.*
10. *Convert JK flip-flop into SR flip-flop.*
11. *Convert D flip-flop into SR flip-flop.*

5.7 Two Marks Questions with Answers

Q.1 *Give the excitation table of SR flip-flop.* **Dec.-03**

Ans. : Refer section 5.5.3.1.

Q.2 *With reference to a JK flip-flop, what is racing ?* **May-04, Dec.-08**

Ans. : In a level triggered J-K flip-flop, when J and K are both high, then the output toggles continuously. This condition is called a race around condition.

Q.3 *Differentiate between flip-flop and latch.* **May-05**

Ans. :

Sr. No.	Latch	Flip-flop
1.	A simple latch is the basis for flip-flop building.	Flip-flop is built by connecting some additional components around a latch.
2.	Latch is level triggered either positive level or negative level triggered.	Flip-flop is pulse or clock-edge triggered either positive edge or negative edge triggered.
3.	The latch output responds to inputs, until active level is maintained at the enable input.	Flip-flop output responds to inputs only at the specified (positive or negative) edges of clock pulse.

Q.4 *Give the excitation table for JK flip-flop.* **May-05, 09**

Ans. : Refer section 5.5.3.2.

Q.5 *Obtain the excitation table of D and JK flip-flops.* **May-06**

Ans. : Refer sections 5.5.3.3 and 5.5.3.2.

Q.6 *Give the meaning for edge triggering in flip-flops.* **Dec.-06, 07**

Ans. : In the edge triggering, the output responds to the changes in the input only at the positive or negative edge of the clock pulse at the clock input. There are two types of edge triggering.

- **Positive edge triggering :** Here, the output responds to the changes in the input only at the positive edge of the clock pulse at the clock input.
- **Negative edge triggering :** Here, the output responds to the changes in the input only at the negative edge of the clock pulse at the clock input.

Q.7 *If the input frequency of a T FF is 1600 kHz, what will be the output frequency ? Give reason for your answer.* **Dec.-06**

Ans. : 800 kHz, because it toggles at every clock pulse.

Q.8 *Convert JK FF to D FF.* **May-07**

Ans. : Refer section 5.6.5.

Q.9 *Define a sequential logic circuit. Give an example.* **May-08**

Ans. : The circuits in which the output variables depend not only on the present input but they also depend upon the past history of these input variables are known as sequential logic circuits. Flip-flops, counters, registers are the examples of sequential logic circuit.

Q.10 *Draw the logic circuit of a clocked JK flip-flop.* **May-08**

Ans. : Refer section 5.4.5.

Q.11 *Differentiate between combinational and sequential circuits.* **Dec.-08**

Ans. : Refer section 5.1.1.

Q.12 *What are synchronous sequential circuits ?* **May-10**

Ans. : Synchronous sequential circuits are those in which signals can affect the memory elements only at discrete instants of time. Clocked flip-flops are examples of synchronous sequential circuits.

Q.13 *Draw the circuit of SR flip-flop.* **May-10**

Ans. : Refer section 5.4.3.

Q.14 *Give the characteristic equation and state diagram of JK flip-flop.*

May-07, 10

Ans. :

- **Characteristic equation :** $Q_{n+1} = J\,\overline{Q}_n + K\,Q_n$
- **State diagram :**

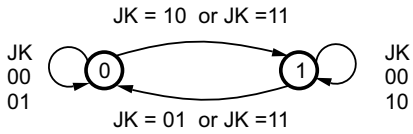


Fig. 5.7.1

Q.15 *What is the operation of D flip-flop ?*

Ans. : In D flip-flop during the occurrence of clock pulse if D = 1, the output Q is set and if D = 0, the output is reset.

Q.16 *What is the operation of JK flip-flop ?*

Ans. :

- When K input is low and J input is high the Q output of flip-flop is set.
- When K input is high and J input is low the Q output of flip-flop is reset.
- When both the inputs K and J are low the output does not change.
- When both the inputs K and J are high the output toggle on the next positive clock edge.

Q.17 ***What is the operation of T flip-flop ?***

Ans. : T flip-flop is also known as Toggle flip-flop.

- When T = 0 there is no change in the output.
- When T = 1 the output switch to the complement state (i.e.) the output toggles.

Q.18 ***What is a master-slave flip-flop ?***

Ans. : A master-slave flip-flop consists of two flip-flops where one circuit serves as a master and the other as a slave. The output of the master flip-flop is fed as an input to the slave flip-flop. The master flip-flop is triggered at the positive edge of the clock and slave flip-flop is triggered at the negative edge of the clock.

Q.19 ***Differentiate between edge triggering and level triggering.***

Ans. : Refer section 5.4.2.

❑❑❑

Notes

6 Counters

Syllabus

Asynchronous ripple or serial counter - Asynchronous up/down counter - Synchronous counters - Synchronous up/down counters - Programmable counters - Design of synchronous counters-Modulo-n counter

Contents

6.1 Introduction

*6.2 Ripple / Asynchronous Counters . . . . . **Dec.-09, May-03,** · · · · · · · · Marks 16*

*6.3 Design of Ripple (Asynchronous) Counters . . . . . **Dec.-04, May-06, 09, 11,** · · Marks 16*

*6.4 Synchronous Counters . . . . . **Dec.-06, May-08,** · · · · · · · · · Marks 8*

*6.5 Design of Synchronous Counters **May-03, 04, 05, 10, 12,***

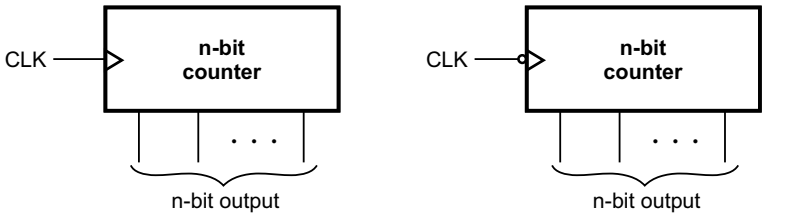
*. . . . . **Dec.-03, 07, 09, 11,** · · · · · · Marks 16*

6.6 Two Marks Questions with Answers

6.1 Introduction

A group of flip-flops connected together forms a **register.** A register is used solely for storing and shifting data which is in the form of 1s and/or 0s, entered from an external source. It has no specific sequence of states except in certain very specialized applications. A **counter** is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. On arrival of each clock pulse, the counter is incremented by one. In case of down counter, it is decremented by one.

The Fig. 6.1.1 shows the logic symbol of a binary counter. External clock is applied to the clock input of the counter. The counter can be positive edge triggered or negative edge triggered. The n-bit binary counter has n flip-flops and it has 2^n distinct states of outputs. For example, 2-bit counter has 2 flip-flops and it has $4(2^2)$ distinct states : 00, 01, 10 and 11. Similarly, the 3-bit binary counter has 3 flip-flops and it has $8(2^3)$ distinct states : 000, 001, 010, 011, 100, 101 110 and 111.



(a) Positive edge triggered n-bit counter **(b) Negative edge triggered n-bit counter**

Fig. 6.1.1 Logic symbol of counter

The maximum count that the binary counter can count is 2^{n-1}. For example, in 2-bit binary counter, the maximum count is $2^2 - 1 = 3$ (11 in binary). After reaching the maximum count the counter resets to 0 on arrival of the next clock pulse and it starts counting again.

Synchronous Counter : When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called synchronous counter.

Asynchronous Counter / Ripple Counter : A binary asynchronous/ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming clock pulses.

The Table 6.1.1 shows the comparison between synchronous and asynchronous counters.

Sr. No.	Asynchronous counters	Synchronous counters
1.	In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2.	All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
3.	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are high speed counters and are preferred when number of flip-flops increases in the given design.

Table 6.1.1 Synchronous Vs asynchronous counters

Modulus of Counter

The total number of counts or stable states a counter can indicate is called 'Modulus'. For instance, the modulus of a four-stage counter would be 16_{10}, since it is capable of indicating 0000_2 to 1111_2. The term 'modulo' is used to describe the count capability of counters. For example, mod 6 counter goes through states 0 to 5 and mod 4 counter goes through states 0 - 3.

Example 6.1.1 *Draw the state diagram of MOD-10 counter.* **Dec.-08, Marks 2**

Solution :

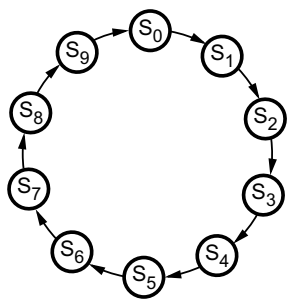


Fig. 6.1.2

Review Questions

1. *What is counter ?*
2. *State types of counters.*
3. *Compare synchronous and asynchronous counter.*
4. *Define modulus of a counter.*

6.2 Ripple / Asynchronous Counters

May-03, Dec.-09

A binary ripple/asynchronous counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming clock pulses. A complementing flip-flops can be obtained from a JK flip-flop with the J and K inputs tied together as shown in the Fig. 6.1.2 or from a T flip-flop. A third alternative is to use a D flip-flop with the complement output connected to the D input. In this way, the D input is always the complement of the present state and the next clock pulse will cause the flip-flop to complement. Let us see the ripple counter using JK flip-flop.

Fig. 6.2.1 shows 2-bit asynchronous counter using JK flip-flops. As shown in Fig. 6.2.1, the clock signal is connected to the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the Q_A output of the first stage. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the Q_A output of first stage can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

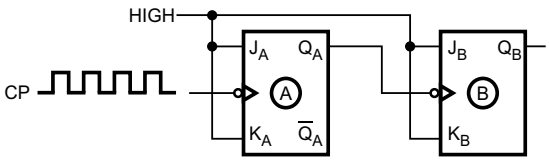


Fig. 6.2.1 A two-bit asynchronous binary counter

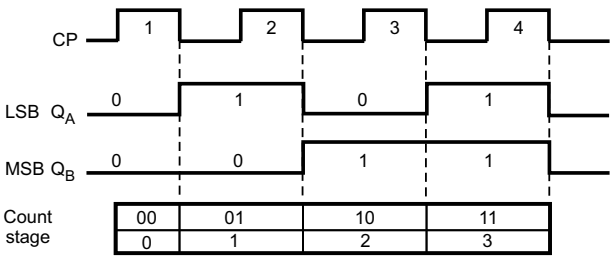


Fig. 6.2.1 (a) Timing diagram for the counter of Fig. 6.2.1

Fig. 6.2.1 (a) shows the timing diagram for two-bit asynchronous counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock. J and K input of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

Example 6.2.1 *Extend the counter shown in Fig. 6.2.1 for 3-stages, and draw output waveforms.*

Solution :

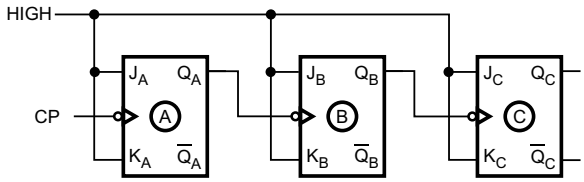


Fig. 6.2.2 (a) Logic diagram

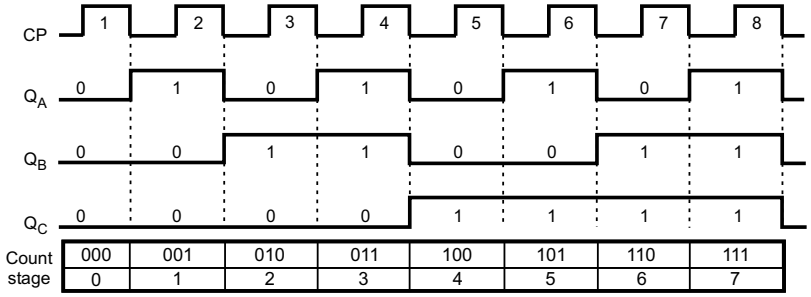


Fig. 6.2.2 (b) Output waveforms for 3-bit asynchronous counter

In Fig. 6.2.2 (b), timing diagram for 3-bit asynchronous counter we have not considered the propagation delays of flip-flops, for simplicity. If we consider the propagation delays of flip-flops we get timing diagram as shown in Fig. 6.2.3.

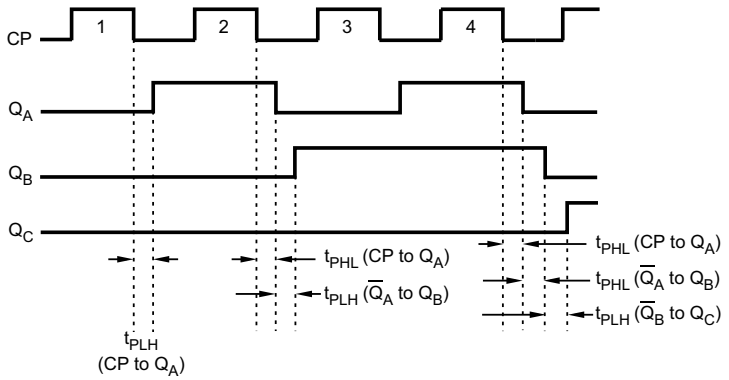


Fig. 6.2.3 Propagation delays in a ripple clocked binary counter

The timing diagram shows propagation delays. We can see that propagation delay of the first stage is added in the propagation delay of second stage to decide the transition time for third stage. This cumulative delay of an asynchronous counter is a major disadvantage in many applications because it limits the rate at which the counter can be clocked and creates decoding problems.

Example 6.2.2 *Draw the logic diagram for 3-stage asynchronous counter with negative edge triggered flip-flops.*

Solution : When flip-flops are negatively edge triggered, the Q output of previous stage is connected to the clock input of the next stage. Fig. 6.2.4 shows 3-stage asynchronous counter with negative edge triggered flip-flops.

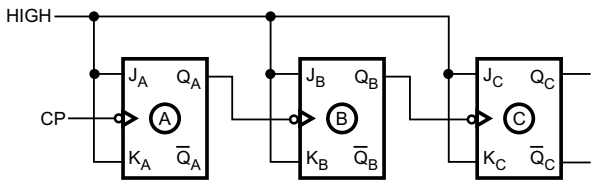


Fig. 6.2.4 Logic diagram of 3-stage negative edge triggered counter

Example 6.2.3 *A counter has 14 stable states 0000 through 1101. If the input frequency is 50 kHz what will be its output frequency?*

Solution : $\frac{50\,\text{kHz}}{14} = 3.57\ \text{kHz}$

Example 6.2.4 *The t_{pd} for each flip-flop is 50 ns , determine the maximum operating frequency for MOD-32 ripple counter.*

Solution : We know that MOD-32 uses five flip-flops. With t_{pd} = 50 ns, the f_{max} for ripple counter can be given as,

$$f_{max\ (ripple)} = \frac{1}{5 \times 50\,ns} = 4\ \text{MHz}$$

6.2.1 Asynchronous / Ripple Down Counter

In the last section we have seen that the output of counter is incremented by one for each clock transition. Therefore, we call such counters as up **counters.** In this section we see the asynchronous/ripple down counter. The down counter will count downward from a maximum count to zero.

The Fig. 6.2.5 shows the 4-bit asynchronous down counter using JK flip-flops. Here, the clock signal is connected to the clock input of only first flip-flop. This connection is same as asynchronous/ripple up counter. However, the clock input of the remaining flip-flops is triggered by the $\overline{Q}_A$ output of the previous stage instead of Q_A output of the previous stage.

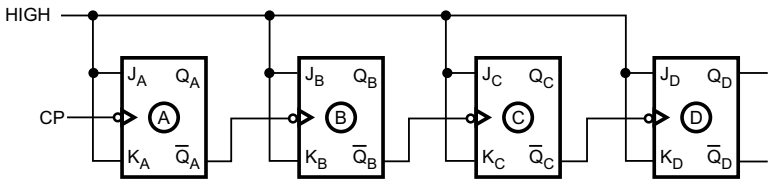


Fig. 6.2.5 4-bit asynchronous down counter

The Fig. 6.2.6 shows the timing diagram for 4-bit asynchronous down counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock. Again the J and K inputs of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

Down counters are not as widely used as up counters. They are used in situations where it must be known when a desired number of input pulses has occurred. In these situations the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses have occurred.

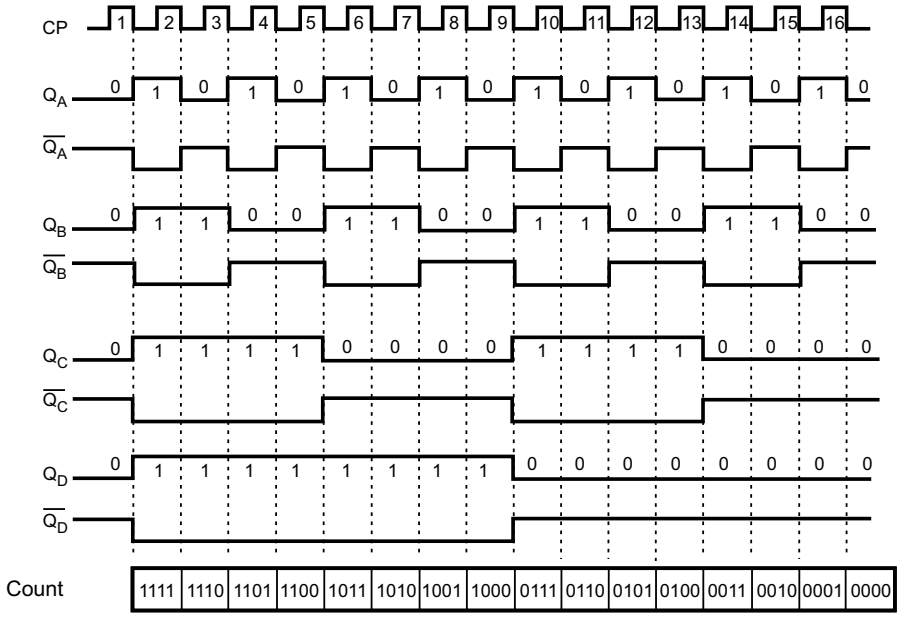


Fig. 6.2.6 Timing diagram of 4-bit asynchronous down counter

Example 6.2.5 *For the ripple counter shown in Fig. 6.2.7, show the complete timing diagram for eight clock pulses, showing the clock, Q_0 and Q_1 waveforms.*

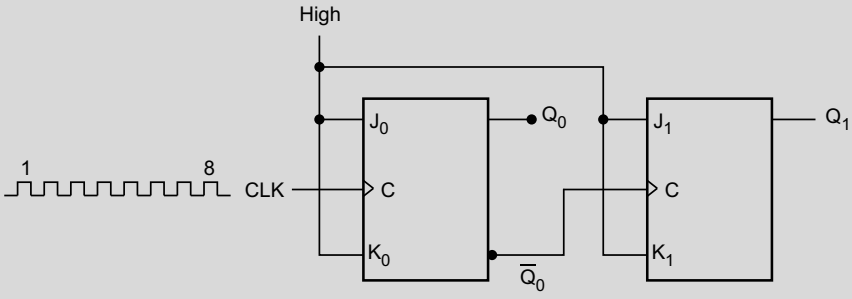


Fig. 6.2.7

Solution :

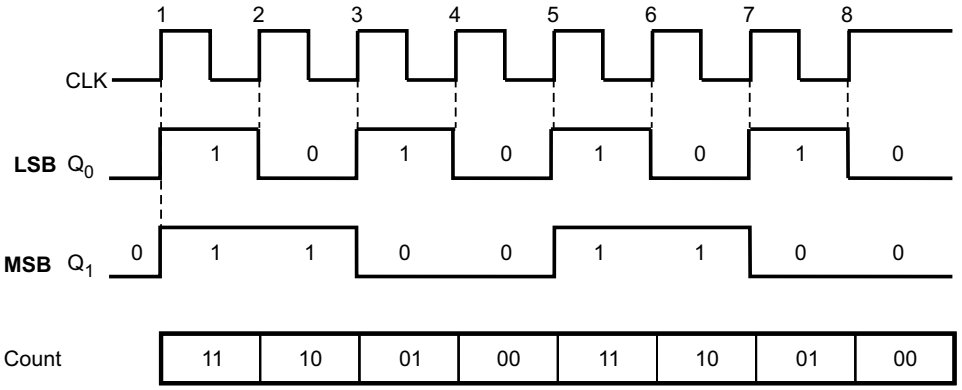
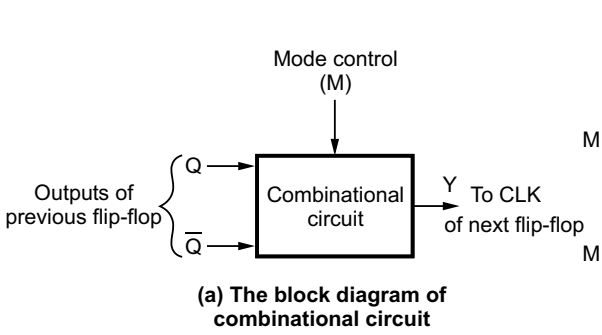


Fig. 6.2.8

6.2.2 Asynchronous Up / Down Counter

To form an asynchronous up/down counter one control input say M is necessary to control the operation of the up/down counter. When M = 0, the counter will count up and when M = 1, the counter will count down. To achieve this the M input should be used to control whether the normal flip-flop output (Q) or the inverted flip-flop output ($\overline{Q}$) is fed to drive the clock signal of the successive stage flip-flop, as shown in Fig. 6.2.9 (a). The truth table for such combinational circuit is shown in Fig. 6.2.9 (b).

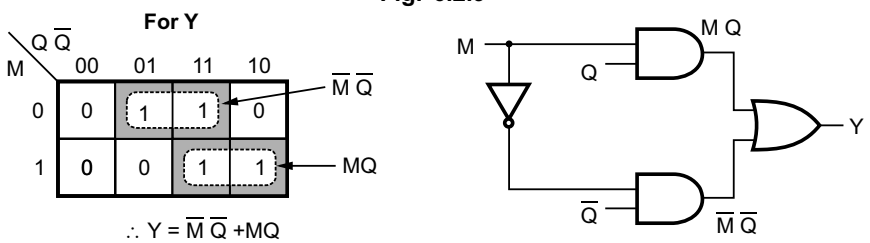


(a) The block diagram of combinational circuit

	Inputs M	Q	$\overline{Q}$	Output Y	
M = 0	0	0	0	0	$Y = \overline{Q}$ for down counting
	0	0	1	1	
	0	1	0	0	
	0	1	1	1	
M = 1	1	0	0	0	Y = Q for up counting
	1	0	1	0	
	1	1	0	1	
	1	1	1	1	

(b) Truth table

Fig. 6.2.9



a) K-map simplification **b) Logic diagram**

Fig. 6.2.10

The Fig. 6.2.11 shows the 3-bit up/down counter that will count from 000 up to 111 when the mode control input M is 1 and from 111 down to 000 when mode control input M is 0.

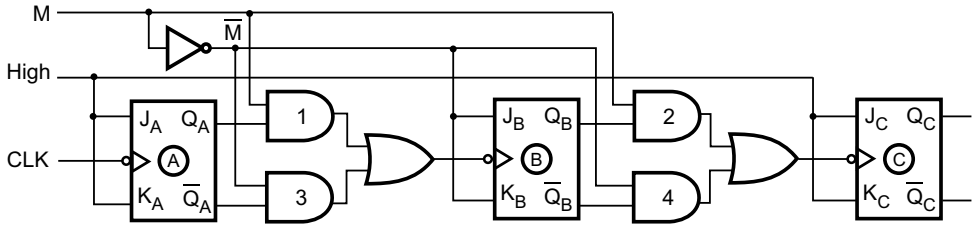


Fig. 6.2.11 3-bit asynchronous up/down counter

A logic 1 on M enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs to drive the clock inputs of their respective next stages. So that counter will count up. When M is logic 0, AND gates 1 and 2 are disabled and AND gate 3 and 4 are enabled. This allows the $\overline{Q}_A$ and $\overline{Q}_B$ outputs to drive the clock inputs of their respective next stages so that counter will count down. The Fig. 6.2.12 shows the timing diagram for 3-bit up/down ripple counter. (See Fig. 6.2.12 on next page).

6.2.3 Decoding Gates

Decoding gates are used to indicate whether counter has reached to particular state. The outputs of the counter are connected to the AND gate as inputs and the output of the AND gate goes high for particular state. Let us see the Fig. 6.2.13 (a). Here, the output of decoding gate goes high when counter outputs are C = 1, B = 1 and A = 1. In Fig. 6.2.13 (b) the output of decoding gate goes high when counter outputs are C = 1, B = 0 and A = 0.

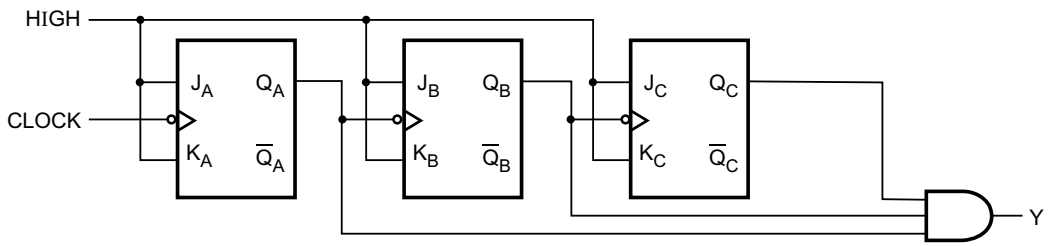


Fig. 6.2.13 (a) Decoding gate to indicate state 7 (111)

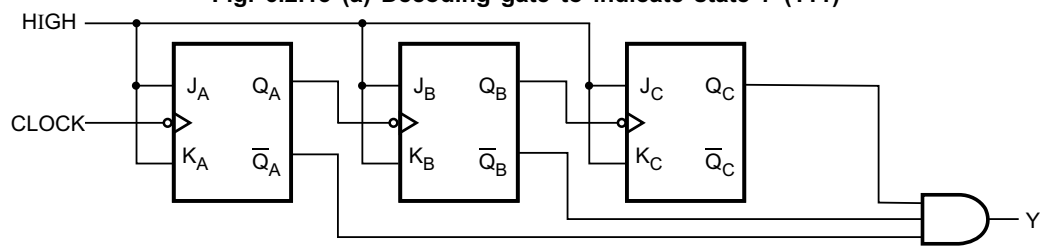


Fig. 6.2.13 (b) Decoding gate to indicate state 4 (100)

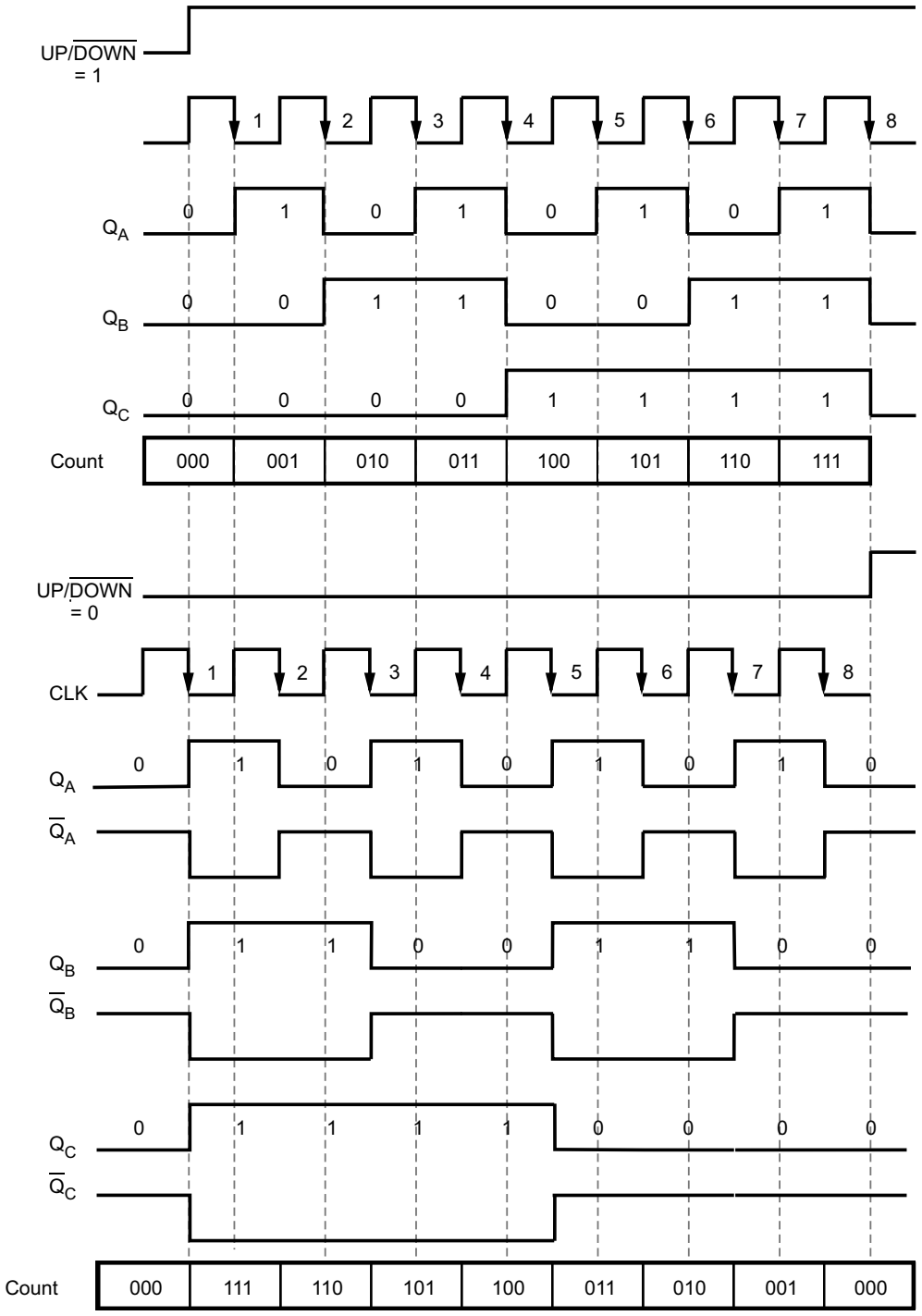


Fig. 6.2.12 Timing diagram for 3-bit UP/ $\overline{\text{DOWN}}$ ripple counter

Similarly, we can connect corresponding outputs to decoding gate inputs to indicate desired state. The Fig. 6.2.14 gives these connections for all possible state detection for 3-bit counter.

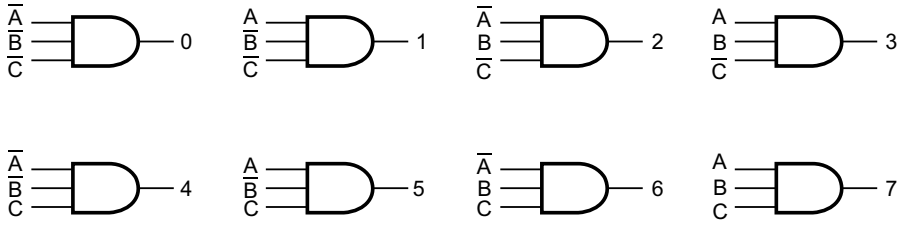

$\overline{A}$ $\overline{B}$ $\overline{C}$ 0
A $\overline{B}$ $\overline{C}$ 1
$\overline{A}$ B $\overline{C}$ 2
A B $\overline{C}$ 3
$\overline{A}$ $\overline{B}$ C 4
A $\overline{B}$ C 5
$\overline{A}$ B C 6
A B C 7

Fig. 6.2.14 Decoding gate connections

6.2.4 Problem Faced by Ripple Counters (Glitch Problem)

We know that, due to the propagation delay the output flip-flop is delayed by time t_p. This illustrated in the waveform shown in Fig. 6.2.15 shows the waveform of the circuit which decodes state 6. Here, the output of flip-flop A triggers the flip-flop B, hence the B waveform is delayed by one flip-flop delay time (t_p) from the negative transition of A. Similarly, the C waveform is delayed by t_p from each negative transition of B.

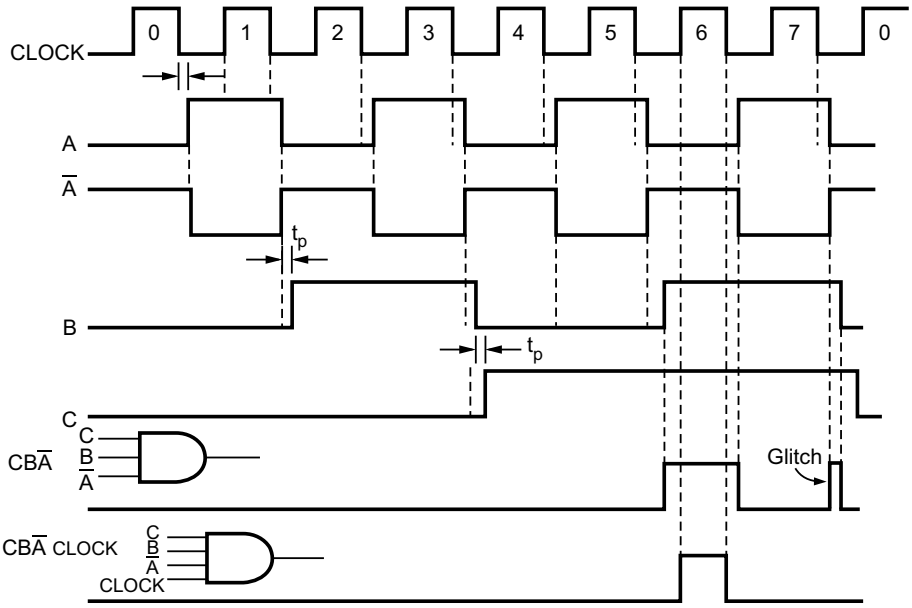


Fig. 6.2.15 Elimination of glitch

Let us observe the output waveform when the counter progresses from state 7 to state 0. At point X, A goes low ($\overline{A}$ goes high); however, because of flip-flop delay time, B does not go low until point Y. Thus between points X and Y we have the condition C = 1, B = 1 and A = 1. As a result, the output is high between points X and Y. This

undesirable output is known as **glitch**. We can avoid the glitch on the output waveform by connecting clock as a fourth input to the decoding gate along with inputs A, B and C. This is illustrated in the Fig. 6.2.15.

Review Questions

1. *Explain in detail the operation of a 4-bit binary ripple counter.* **Dec.-09, Marks 16**
2. *Explain the working of asynchronous down counter.*
3. *Design and explain the working of an up-down ripple counter.* **May-03, Marks 8**
4. *What do you mean by asynchronous counter?*
5. *State the problem faced by ripple counters.*

6.3 Design of Ripple (Asynchronous) Counters **Dec.-04, May-06, 09, 11**

The steps involved in the design of asynchronous counter are :

1. Determine the number of flip-flops needed.
2. Choose the type of flip-flops to be used : T or JK. If T flip-flops are used connect T input of all flip-flops to logic 1. If JK flip-flops are used connect both J and K inputs of all flip flops to logic 1.

   Such connection toggles the flip-flop output on each clock transition.
3. Write the truth table for the counter.
4. Derive the reset logic by K-map simplification.
5. Draw the logic diagram.

Example 6.3.1 *Design BCD ripple counter using JK flip-flop.* **Dec.-04, May-06, 11, Marks 16**

Solution :

Step 1 : Determine the number of flip-flops needed. The BCD counter goes through states 0-9, i.e. total 10 states. Thus, N = 10 and for $2^n \geq N$, we need n = 4, i.e. 4 flip-flops required.

Step 2 : Type of flip-flops to be used : JK

Step 3 : Write the truth table for the counter

CLK	A	B	C	D	Output of reset logic Y	
0	0	0	0	0	1	Valid states
1	0	0	0	1	1	
2	0	0	1	0	1	
3	0	0	1	1	1	
4	0	1	0	0	1	
5	0	1	0	1	1	
6	0	1	1	0	1	
7	0	1	1	1	1	
8	1	0	0	0	1	
9	1	0	0	1	1	
–	1	0	1	0	0	Invalid states
–	1	0	1	1	0	
–	1	1	0	0	0	
–	1	1	0	1	0	
–	1	1	1	0	0	
–	1	1	1	1	0	

Note : The reset input (CLR) of each Flip-Flop is active-low input. By making CLR input of all Flip-Flops logic 0, we can reset the counter. Thus reset logic is designed such a way that for invalid states, Y = 0 and counter resets.

Table 6.3.1 Truth table for BCD counter

Step 4 : Derive reset logic

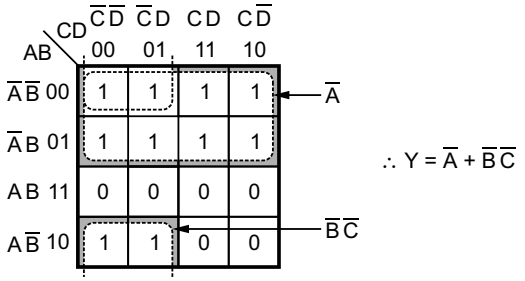


Fig. 6.3.1

Step 5 : Draw logic diagram.

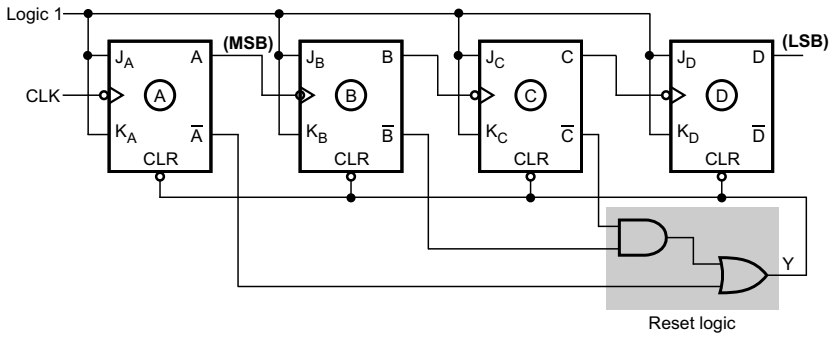


Fig. 6.3.2 Logic diagram of BCD ripple counter

Example 6.3.2 *Design a 3-bit asynchronous ripple counter using T flip-flops and explain its operation.* **May-09, Marks 16**

Solution : The Fig. 6.3.3 shows a 3-bit asynchronous ripple counter using T flip-flops. As shown in Fig. 6.3.3, the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the Q_A output of the first stage and third stage flip-flop is triggered by the Q_A output of second stage. Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the Q_A output of previous stage can never occur at exactly the same time. Therefore, flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

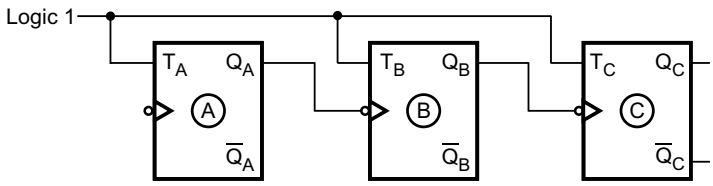


Fig. 6.3.3 3-bit asynchronous counter using T flip-flops

Since, T input is connected to logic 1 each flip-flop toggles at clock input. The Fig. 6.3.4 shows the timing diagram for 3-bit asynchronous counter.

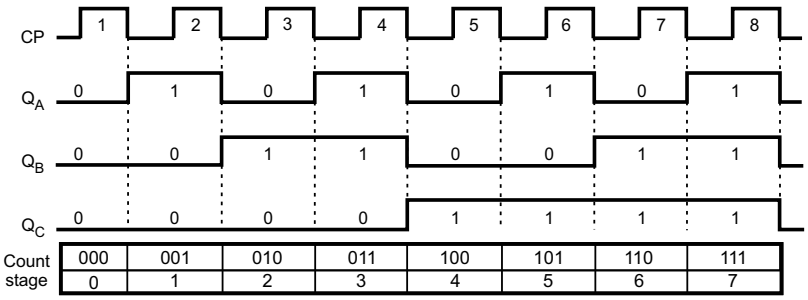


Fig. 6.3.4 Output waveforms for 3-bit asynchronous counter

Example 6.3.3 *Design mod 6 ripple counter using T flip-flops.*

Solution :

Step 1 : Determine the number of flip-flop required. Here, counter goes through 0 - 5 states, i.e., total 6 states. Thus N = 6 and for $2^n \geq N$ we need n = 3, i.e. 3 flip-flops.

Step 2 : Type of flip-flops to be used : T

Step 3 : Write the truth table for counter

CLK	A	B	C	Output of reset logic Y
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
–	1	1	0	0
–	1	1	1	0

Valid states

Invalid states

Fig. 6.3.5

Step 4 : Derive reset logic

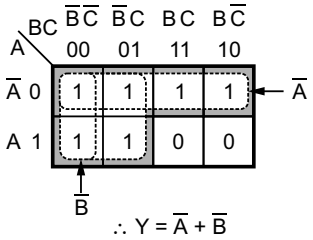


Fig. 6.3.6

Step 5 : Draw logic diagram

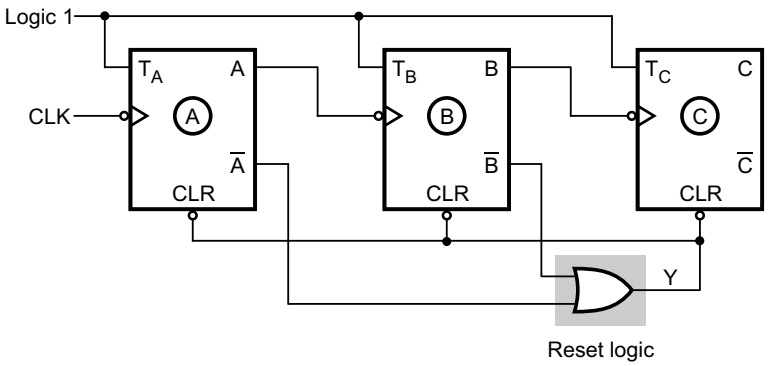


Fig. 6.3.7

Review Questions

1. *Explain the working of BCD ripple counter with timing diagrams.* **Dec.-04, Marks 16**
2. *Design mod 5 ripple counter using T Flip-Flops.*

6.4 Synchronous Counters

Dec.-06, May-08

When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called as synchronous counter.

6.4.1 2-bit Synchronous Binary Up Counter

Fig. 6.4.1 shows two stage synchronous counter.

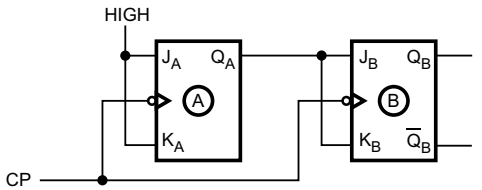
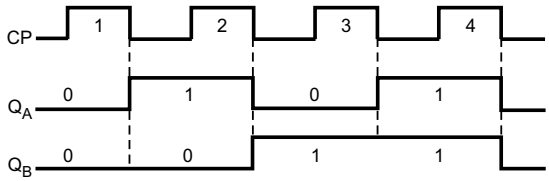


Fig. 6.4.1 A two-bit synchronous binary counter

Here, clock signal is connected in parallel to clock inputs of both the flip-flops. But the Q_A output of first stage is used to drive the J and K inputs of the second stage. Let us see the operation of the circuit. Initially, we assume that the $Q_A = Q_B = 0$. When positive edge of the first clock pulse is applied, flip-flop A will toggle because $J_A = K_A = 1$, whereas flip-flop B output will remain zero because $J_B = K_B = 0$. After first clock pulse $Q_A = 1$ and $Q_B = 0$. At negative going edge of the second clock pulse both flip-flops will toggle because they both have a toggle condition on their J and K inputs ($J_A = K_A = J_B = K_B = 1$). Thus after second clock pulse, $Q_A = 0$ and $Q_B = 1$. At negative going edge of the third clock pulse flip-flop A toggles making $Q_A = 1$, but flip-flop B remains set i.e. $Q_B = 1$. Finally, at the leading edge of the fourth clock pulse both flip-flops toggle as their JK inputs are at logic 1. This results $Q_A = Q_B = 0$ and counter recycled back to its original state. The timing details of above operation is shown in Fig. 6.4.2.



(a)

CP	Q_B	Q_A
0	0	0
1	0	1
2	1	0
3	1	1

(b)

Fig. 6.4.2 Timing diagram and state sequence for the 2-bit synchronous counter

6.4.2 3-bit Synchronous Binary Up Counter

Fig. 6.4.3 (a) shows 3-bit synchronous binary counter and its timing diagram. The state sequence for this counter is shown in Table 6.4.1.

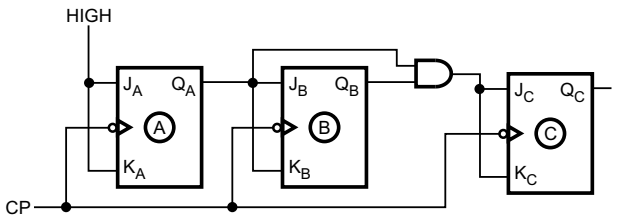


Fig. 6.4.3 (a) A three-bit synchronous binary counter

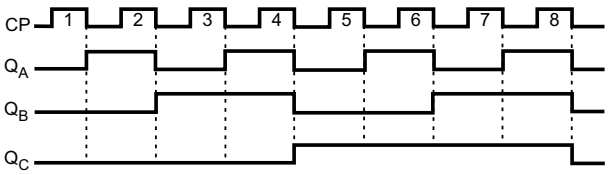


Fig. 6.4.3 (b) Timing diagram for 3-bit synchronous binary counter

Looking at Fig. 6.4.3 (b), we can see that Q_A changes on each clock pulse as we progress from its original state to its final state and then back to its original state. To produce this operation, flip-flop A is held in the toggle mode by connecting J and K inputs to HIGH. Now let us see what flip-flop B does. Flip-flop B toggles, when Q_A is 1. When Q_A is a 0, flip-flop B is in the no-change mode and remains in its present state. Looking at the Table 6.4.1 we can notice that flip-flop C has to change its state only when Q_B and Q_A both are at logic 1. This condition is detected by AND gate and applied to the J and K inputs of flip-flop C. Whenever both Q_A and Q_B are HIGH, the output of the AND gate makes the J and K inputs of flip-flop C HIGH and flip-flop C toggles on the following clock pulse. At all other times, the J and K inputs of flip-flop C are held LOW by the AND gate output and flip-flop does not change state.

CP	Q_C	Q_B	Q_A
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Table 6.4.1 State sequence for 3-bit binary counter

6.4.3 4-bit Synchronous Binary Up Counter

Fig. 6.4.4 (a) shows logic diagram and timing diagram for 4-bit synchronous binary counter. As counter is implemented with negative edge triggered flip-flops, the transitions occur at the negative edge of the clock pulse. In this circuit, first three flip-flops work same as 3-bit counter discussed previously.

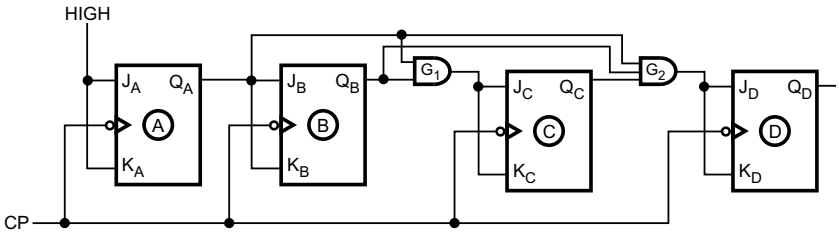


Fig. 6.4.4 (a)

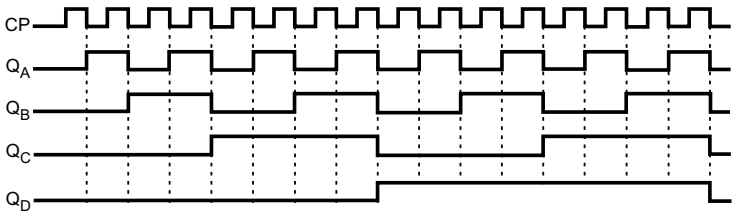


Fig. 6.4.4 (b) A four-bit synchronous binary counter and timing diagram

For the fourth stage, flip-flop has to change the state when $Q_A = Q_B = Q_C = 1$. This condition is decoded by 3-input AND gate G_2. Therefore, when $Q_A = Q_B = Q_C = 1$, flip-flop D toggles and for all other times it is in no change condition.

Example 6.4.1 *Determine f_{max} for the 4-bit synchronous counter if t_{pd} for each flip-flop is 50 ns and t_{pd} for each AND gate is 20 ns. Compare this with f_{max} for a MOD-16 ripple counter.*

Solution : For a synchronous counter the total delay that must be allowed between input clock pulses is equal to flip-flop t_{pd} + AND gate t_{pd}. Thus $T_{clock} \geq 50 + 20 = 70$ ns and so the counter has

$$f_{max} = \frac{1}{70\,ns} = 14.3 \text{ MHz}$$

We know that MOD-16 ripple counter used four flip-flops. With flip-flop $t_{pd} = 50$ ns, the f_{max} for ripple counter can be given as,

$$f_{max\ (ripple)} = \frac{1}{4 \times 50\,ns} = 5 \text{ MHz}$$

6.4.4 Synchronous Down and Up/Down Counters

We have seen that a ripple counter could be made to count down by using the inverted output of each flip-flop to drive the next flip-flops in the counter. A parallel/synchronous down counter can be constructed in a similar manner that is, by using the inverted FF outputs to drive the following JK inputs. For example, the parallel up counter of Fig. 6.4.4 (a) can be converted to a down counter by connecting the $\overline{Q}_A$, $\overline{Q}_B$, $\overline{Q}_C$ and $\overline{Q}_D$ outputs in place of Q_A, Q_B, Q_C and Q_D respectively. The counter will then proceed through the following sequence as input pulses are applied :

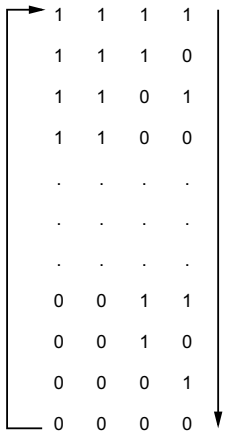


Fig. 6.4.5

To form a parallel up/down counter the control input (UP/$\overline{DOWN}$) is used to control whether the normal flip-flop outputs or the inverted flip-flop outputs are fed to the J and K inputs of the following flip-flops. The Fig. 6.4.5 shows 3-bit up/down counter that will count from 000 up to 111 when the Up/$\overline{Down}$ control input is 1 and from 111 down to 000 when the Up/$\overline{Down}$ control input is 0.

A logic 1 on the Up/$\overline{Down}$ enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs through to the J and K inputs of the next flip-flops so that the counter will count up as pulses are applied. When Up/$\overline{Down}$ line

is logic 0, AND gates 1 and 2 are disables and AND gates 3 and 4 are enabled. This allows the $\overline{Q}_A$ and $\overline{Q}_B$ outputs through to the J and K inputs of the next flip-flops so that the counter will count down as pulses are applied.

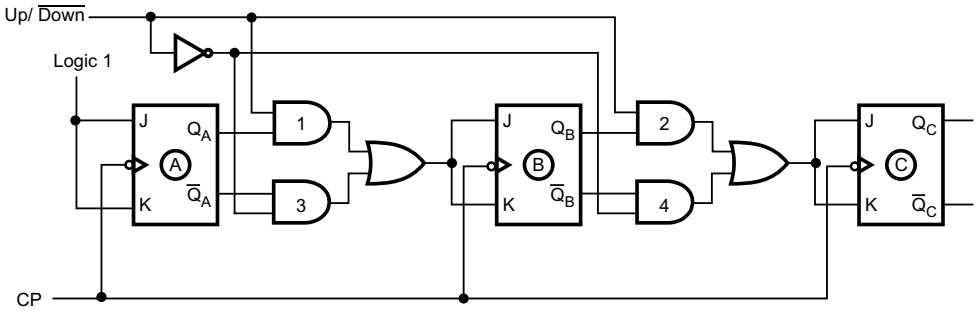


Fig. 6.4.6 3-bit synchronous/parallel up/down counter

6.4.5 Synchronous Vs Asynchronous Counters

The Table 6.4.2 shows the comparison between synchronous and asynchronous counters.

Sr. No.	Asynchronous Counters	Synchronous Counters
1.	In this type of counter flip-flops are connected in such a way that output of first flip-flop drives the clock for the next flip-flop.	In this type there is no connection between output of first flip-flop and clock input of the next flip-flop.
2.	All the flip-flops are not clocked simultaneously.	All the flip-flops are clocked simultaneously.
3.	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of states increases.
4.	Main drawback of these counters is their low speed as the clock is propagated through number of flip-flops before it reaches last flip-flop.	As clock is simultaneously given to all flip-flops there is no problem of propagation delay. Hence they are high speed counters and are preferred when number of flip-flops increases in the given design.

Table 6.4.2 Synchronous Vs asynchronous counters

Review Questions

1. *Define synchronous counter.* **Dec.-06, Marks 2**
2. *Explain the working of 3-bit synchronous binary up counter.* **May-08, Marks 8**
3. *Explain the working of synchronous up/down counter.*
4. *Give the comparison between asynchronous and synchronous counters.*

6.5 Design of Synchronous Counters

May-03, 04, 05, 10, 12, Dec.-03, 07, 09, 11

1. Determine the number of flip-flops needed. If n represents number of flip-flops $2^n \geq$ number of states in the counter.
2. Choose the type of flip-flops to be used.
3. Using excitation table for selected flip-flop determine the excitation table for the counter.
4. Use K-map or any other simplification method to derive the flip-flop input functions.
5. Draw the logic diagram.

Example 6.5.1 *Design a MOD-5 synchronous counter using JK flip-flops and implement it.*

Solution :

Step 1 : Determine the number of flip-flop needed

Flip-flops required are

$$2^n \geq N$$

Here $N = 5$ $\therefore$ $n = 3$ i.e. three flip-flops are required.

Step 2 : Type of flip-flop to be used : JK

Step 3 : Determine the excitation table for the counter.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 6.5.1 Excitation table for JK flip-flop

Present State			Next State			Flip-flop Inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

Table 6.5.2 Excitation table for counter

Step 4 : K-Map simplification

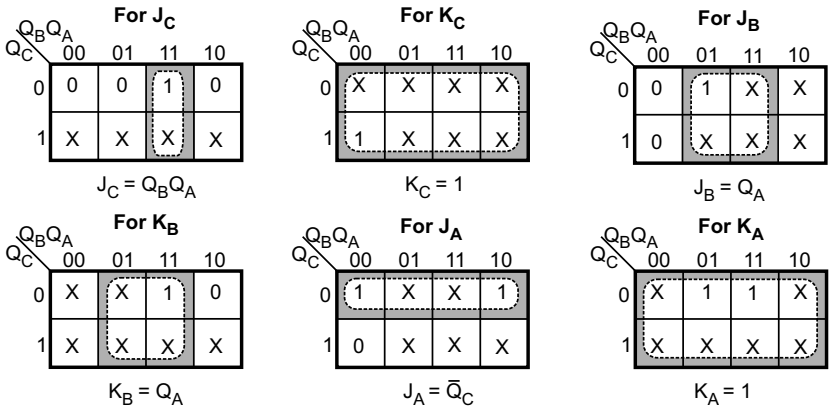


Fig. 6.5.1

Step 5 : Draw the logic diagram

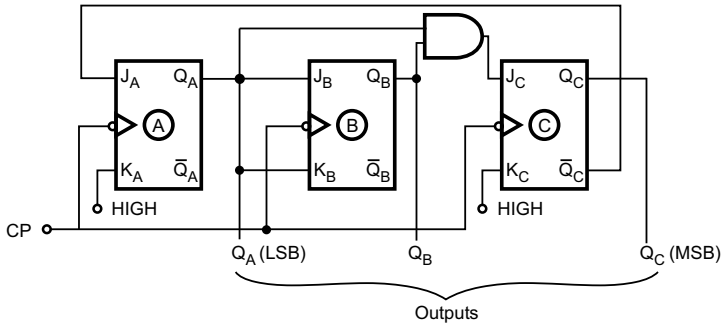


Fig. 6.5.2 MOD-5 synchronous counter

Example 6.5.2 *Design divide by 6 counter using T-flip-flops. Write state table and reduce the expression using K-map.*

Solution :

Step 1 : Determine the number of flip-flops needed.

For designing mod 6 counter using the formula

$$2^n \geq N$$

Here $N = 6 \therefore n = 3$ i.e. 3 flip-flops are required.

Step 2 : Type of flip-flops to be used : T

Step 3 : Determine the excitation table for counter.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 6.5.3 Excitation table for T-flip-flop

CP	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	1
2	0	1	0	0	1	1	0	0	1
3	0	1	1	1	0	0	1	1	1
4	1	0	0	1	0	1	0	0	1
5	1	0	1	0	0	0	1	0	1

Table 6.5.4 Excitation table for counter

Step 4 : K-map simplification.

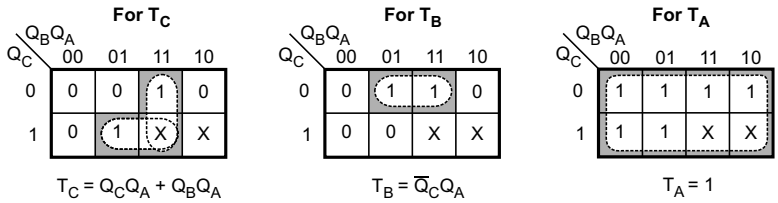


Fig. 6.5.3

Step 5 : Draw the logic diagram.

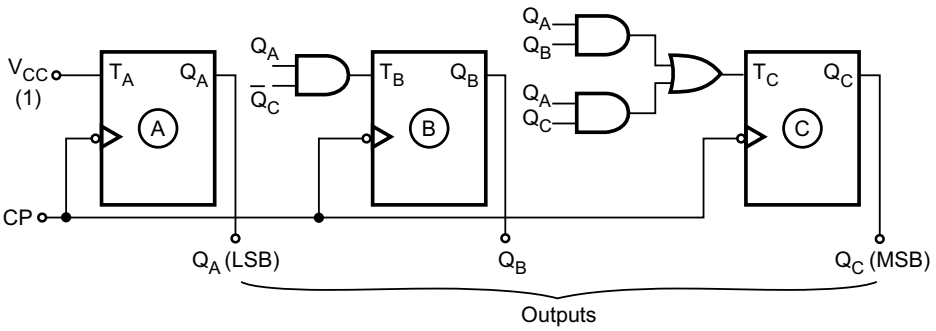


Fig. 6.5.4 Logic diagram

Example 6.5.3 *Using positive edge triggering SR flip-flops design a counter which counts in the following sequence :*
000, 111, 110, 101, 100, 011, 010, 001, 000, ...

Solution :

Step 1 : Determine the number of flip-flops needed

We know that $2^n \geq N$. Here, $N = 8 \quad \therefore \quad n = 3$

Step 2 : Type of flip-flop to be used : SR

Step 3 : Determine the excitation table for counter.

Here, the next state for each present state is written according to given sequence. For example, the next state for the present state 000 is 111.

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Table 6.5.5 Excitation table of SR flip-flop

Present state			Next state			Flip-flop inputs					
A	B	C	A^+	B^+	C^+	S_A	R_A	S_B	R_B	S_C	R_C
0	0	0	1	1	1	1	0	1	0	1	0
0	0	1	0	0	0	0	x	0	x	0	1
0	1	0	0	0	1	0	x	0	1	1	0
0	1	1	0	1	0	0	x	x	0	0	1
1	0	0	0	1	1	0	1	1	0	1	0
1	0	1	1	0	0	x	0	0	x	0	1
1	1	0	1	0	1	x	0	0	1	1	0
1	1	1	1	1	0	x	0	x	0	0	1

Table 6.5.6 Excitation table for counter

Step 4 : K-map simplification.

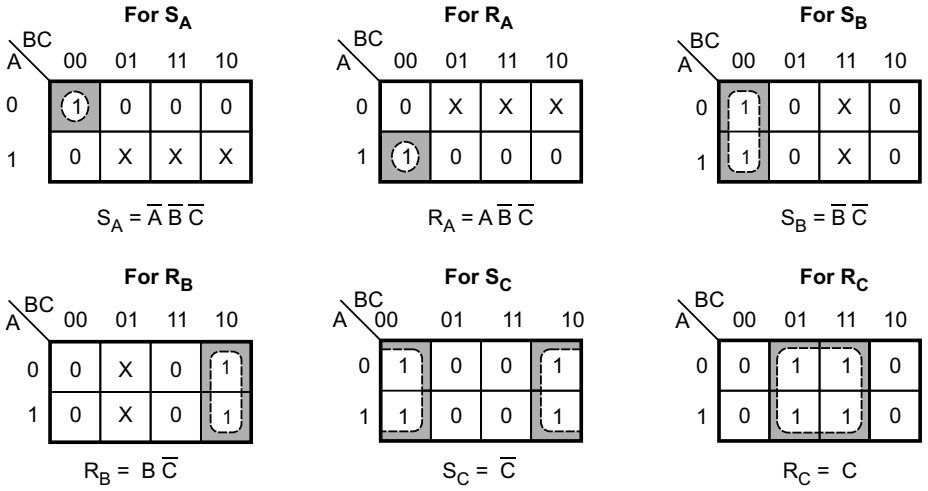


Fig. 6.5.5 (a)

Step 5 : Draw logic diagram

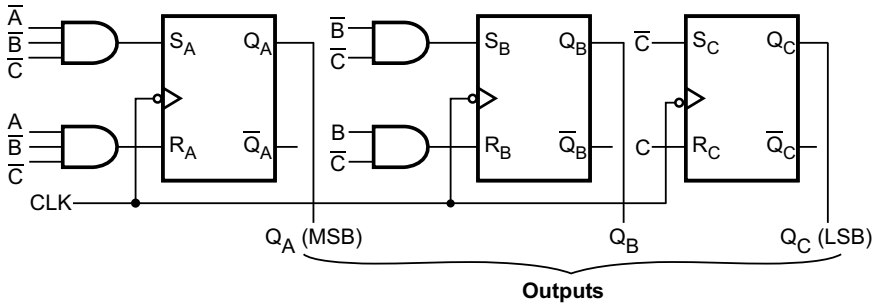


Fig. 6.5.5 (b) Logic diagram

Example 6.5.4 *Design a synchronous decade counter using D flip-flop.*

Solution : The decade counter is a mod-10 counter. It has ten states : 0 - 9.

Step 1 : Determine the number of flip-flops needed.

We know that $2^n \geq N$. Here, N = 10 ∴ n = 4 i.e. 4 flip-flops needed.

Step 2 : Types of flip-flops to be used : D

Step 3 : Determine the excitation table for counter.

Present state				Next state			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0

Table 6.5.7 Excitation table for counter

Step 4 : K-map simplification

$$Q_{D+1} = Q_D\overline{Q}_A + Q_C Q_B Q_A$$

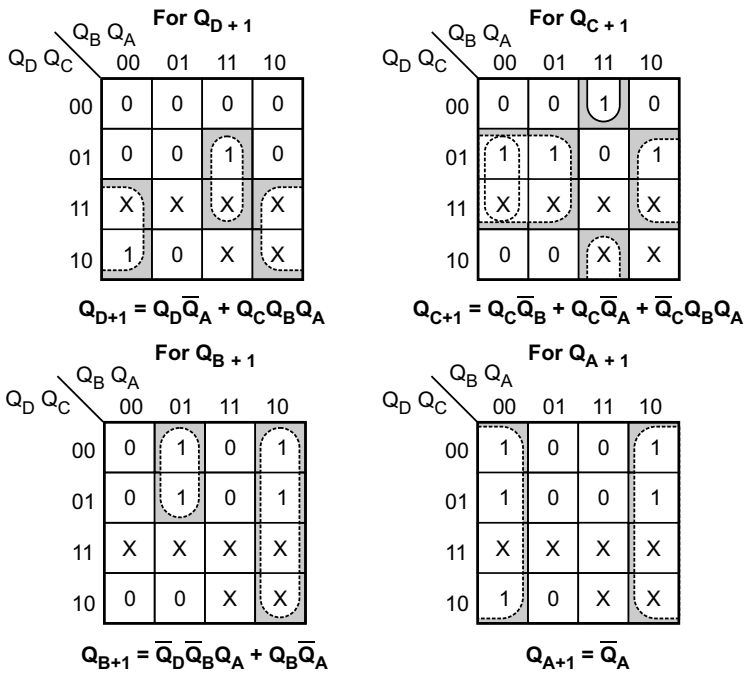


Fig. 6.5.6

$$Q_{D+1} = Q_D\overline{Q}_A + Q_CQ_BQ_A$$
$$Q_{C+1} = Q_C\overline{Q}_B + Q_C\overline{Q}_A + \overline{Q}_CQ_BQ_A$$
$$Q_{B+1} = \overline{Q}_D\overline{Q}_BQ_A + Q_B\overline{Q}_A$$
$$Q_{A+1} = \overline{Q}_A$$

Step 5 : Draw the logic diagram.

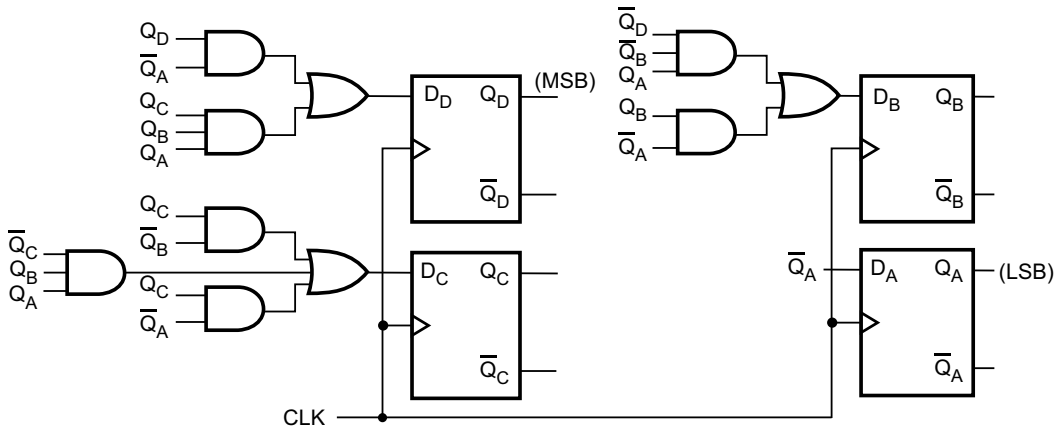


Fig. 6.5.7 Logic diagram

Example 6.5.5 *Design a counter with the sequence 0, 1, 3, 7, 6, 4, 0.*

Solution :

Step 1 : Determine the number of flip-flops needed. Here, counter should count maximum count = 7 = $(111)_2$ which is 3-bit. Thus, we need 3-flip-flops.

Step 2 : Flip-flops to be used : JK.

Step 3 : Determine the excitation table for counter. Here, the next state for each present state is written according to given sequence. For example, the next state for the present state 3 (011) is 7 (111). The counts which are not in sequence are treated as don't cares.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 6.5.8 Excitation table of JK flip-flop

Present state			Next state			Flip-flop Inputs					
A	B	C	A^+	B^+	C^+	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	1	1	1	1	X	X	0	X	0
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	1	0	0	X	0	X	1	0	X
1	1	1	1	1	0	X	0	X	0	X	1

Table 6.5.9 Excitation table for counter

Step 4 : K-map simplification

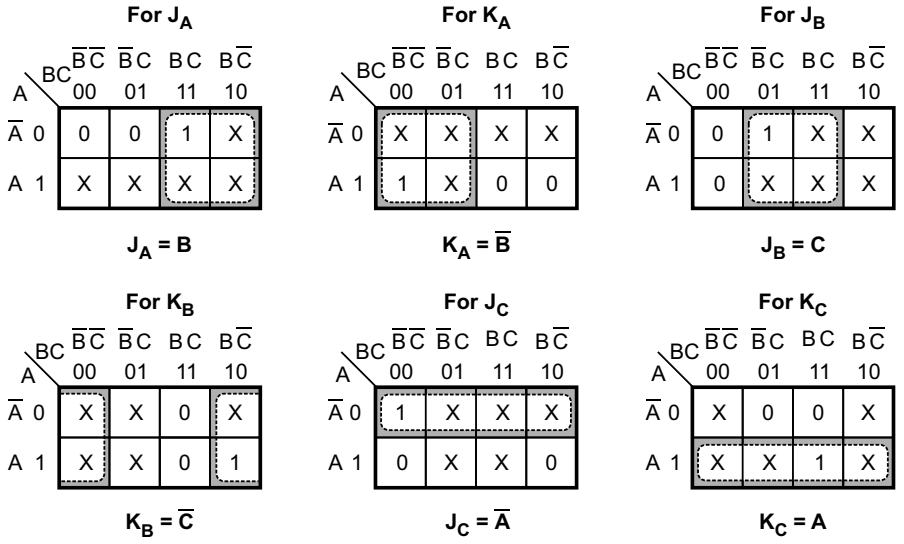


Fig. 6.5.8

Step 5 : Draw logic diagram.

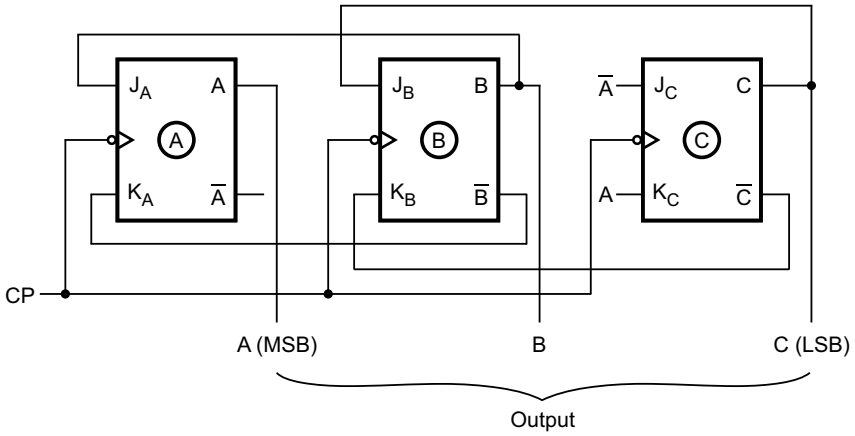


Fig. 6.5.9

Example 6.5.6 *Design a BCD up/down counter using SR flip-flops.*

Solution :

Step 1 : Number of flip-flops needed = 4

Step 2 : Flip-flops to be used = SR

Step 3 : Excitation table for counter

I/P	Present state				Next state				Flip-flop inputs							
$U/\overline{D}$	A	B	C	D	A+	B+	C+	D+	S_A	R_A	S_B	R_B	S_C	R_C	S_D	R_D
0	0	0	0	0	1	0	0	1	1	0	0	×	0	×	1	0
0	0	0	0	1	0	0	0	0	0	×	0	×	0	×	0	1
0	0	0	1	0	0	0	0	1	0	×	0	×	0	1	1	0
0	0	0	1	1	0	0	1	0	0	×	0	×	×	0	0	1
0	0	1	0	0	0	0	1	1	0	×	0	1	1	0	1	0
0	0	1	0	1	0	1	0	0	0	×	×	0	0	×	0	1
0	0	1	1	0	0	1	0	1	0	×	×	0	0	1	1	0
0	0	1	1	1	0	1	1	0	0	×	×	0	×	0	0	1
0	1	0	0	0	0	1	1	1	0	1	1	0	1	0	1	0

0	1	0	0	1	1	0	0	0	×	0	0	×	0	×	0	1
1	0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	0
1	0	0	0	1	0	0	1	0	0	×	0	×	1	0	0	1
1	0	0	1	0	0	0	1	1	0	×	0	×	×	0	1	0
1	0	0	1	1	0	1	0	0	0	×	1	0	0	1	0	1
1	0	1	0	0	0	1	0	1	0	×	×	0	0	×	1	0
1	0	1	0	1	0	1	1	0	0	×	×	0	1	0	0	1
1	0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	0
1	0	1	1	1	1	0	0	0	1	0	0	1	0	1	0	1
1	1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	0
1	1	0	0	1	0	0	0	0	0	1	0	×	0	×	0	1

Table 6.5.10

Step 4 : K-map simplification

For S_A

M = U / $\overline{D}$ =0

AB \ CD	00	01	11	10
00	1	0	0	0
01	0	0	0	0
11	X	X	X	X
10	0	X	X	X

M = U / $\overline{D}$ =1

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	X	0	X	X

$S_A = \overline{M}\,\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + MBCD$

For R_A

M = 0

AB \ CD	00	01	11	10
00	0	X	X	X
01	X	X	X	X
11	X	X	X	X
10	1	0	X	X

M = 1

AB \ CD	00	01	11	10
00	X	X	X	X
01	X	X	0	X
11	X	X	X	X
10	0	1	X	X

$R_A = \overline{M}\,A\,\overline{D} + M\,A\,D$

For S_B

M = 0

AB \ CD	00	01	11	10
00	0	0	0	0
01	0	X	X	X
11	X	X	X	X
10	1	0	X	X

M = 1

AB \ CD	00	01	11	10
00	0	0	1	0
01	X	X	0	X
11	X	X	X	X
10	0	0	X	X

$S_B = \overline{M}\,A\,\overline{D} + M\,\overline{B}\,C\,D$

For R_B

M = 0

AB \ CD	00	01	11	10
00	X	X	X	X
01	1	0	0	0
11	X	X	X	X
10	0	X	X	X

M = 1

AB \ CD	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$R_B = \overline{M}\,B\,\overline{C}\,\overline{D} + MBCD$

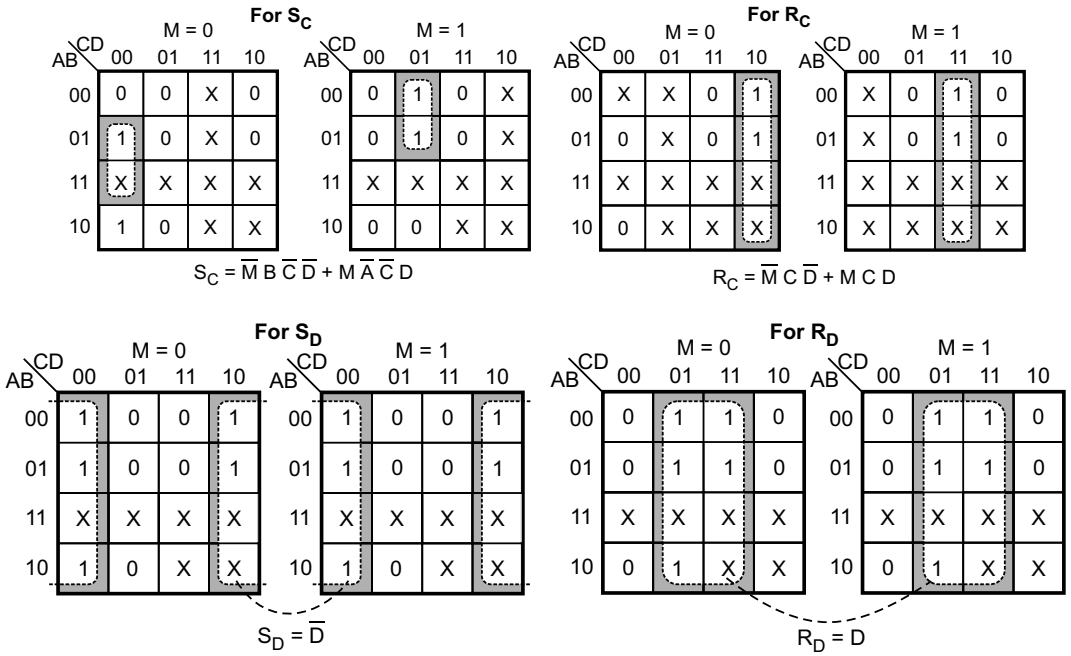


Fig. 6.5.10

Step 5 : Logic diagram

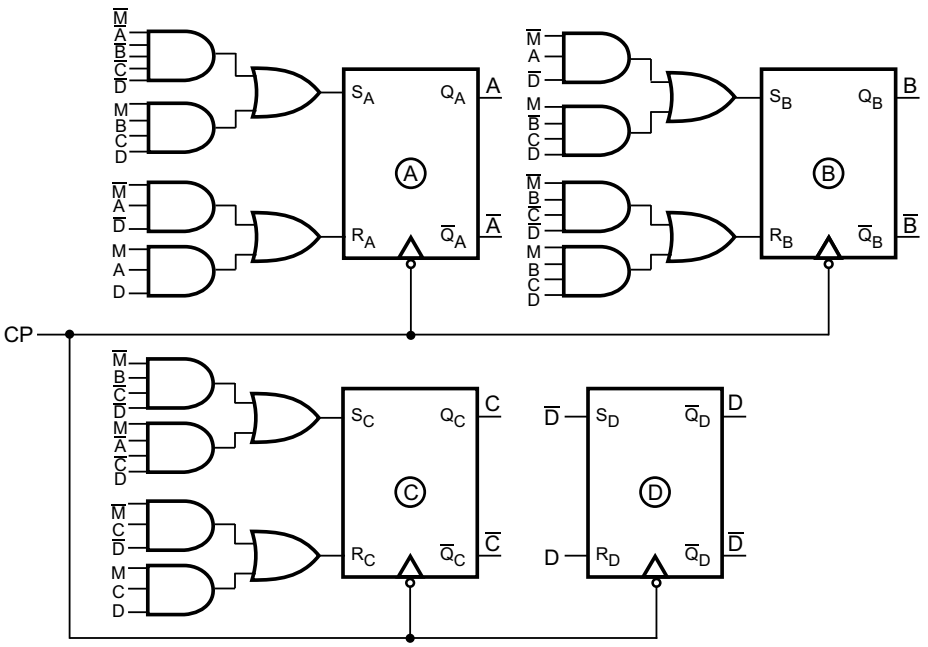


Fig. 6.5.11

Example 6.5.7 *Design a synchronous counter using JK flip-flop to count the following sequence 7, 4, 3, 1, 6, 0, 7 ……*

Solution :

Step 1 : Since $2^3 > 7$, three flip-flops are required

Step 2 : Flip-flops to be used : JK

Step 3 : Excitation table for counter

Present state			Next state			Flip-Flop Inputs					
A	B	C	A^+	B^+	C^+	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	1	1	1	1	X	1	X	1	X
0	0	1	1	0	1	1	X	0	X	X	0
0	1	0	X	X	X	X	X	X	X	X	X
0	1	1	0	0	1	0	X	X	1	X	0
1	0	0	0	1	1	X	1	1	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	1	0	0	X	0	X	1	X	1

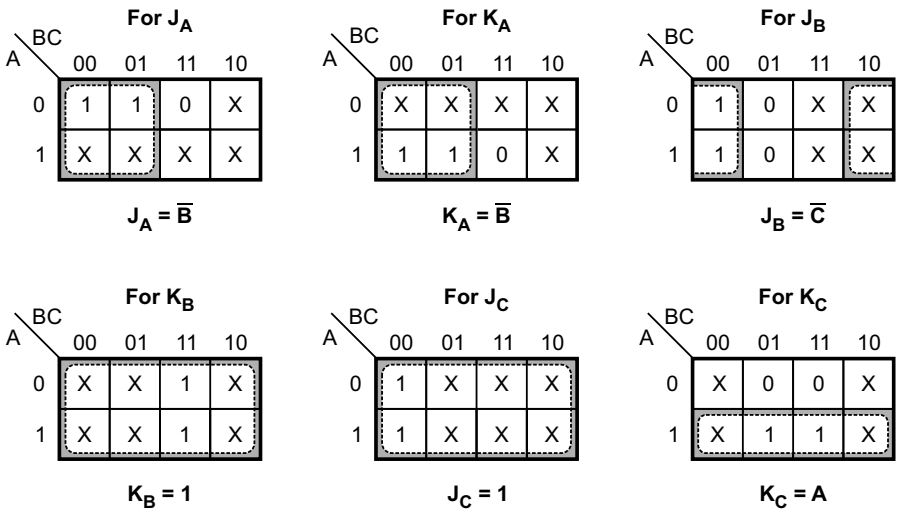


Fig. 6.5.12 (a)

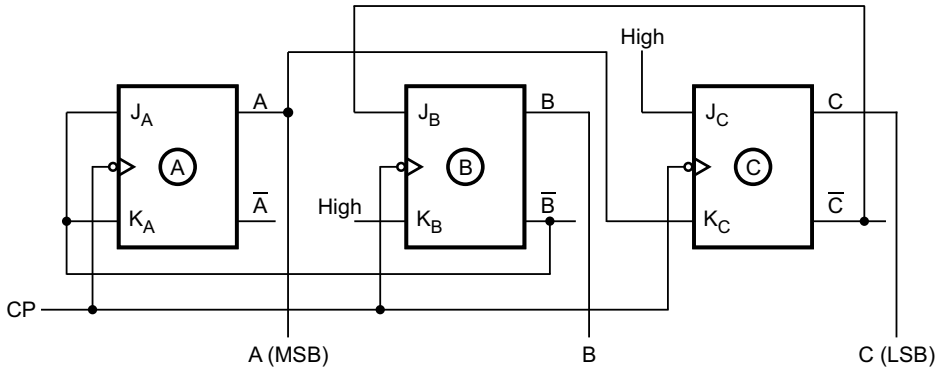


Fig. 6.5.12 (b)

Example 6.5.8 *Design and implement a synchronous decade counter using T flip-flop. Draw the timing diagram.*

Solution :

Step 1 : Since N = 10, n = 4 i.e. flip-flops needed = 4

Step 2 : Flip-flops to be used : T

Step 3 : Determine excitation table for counter

Present state				Next state				Flip-flop Inputs			
Q_D	Q_C	Q_B	Q_A	Q_{D+1}	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_D	T_C	T_B	T_A
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	1
1	0	1	0	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X

Table 6.5.11

Step 4 : K-map simplification

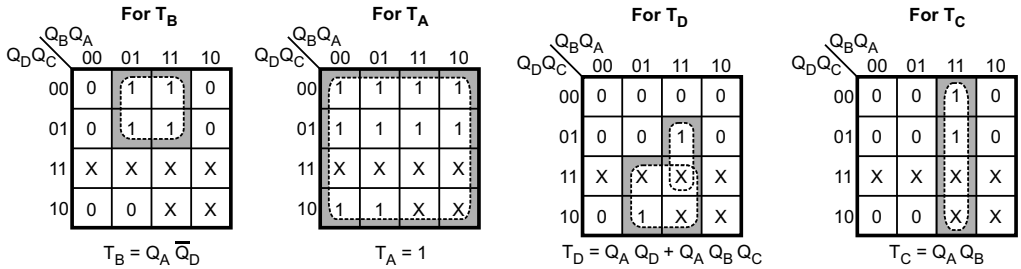


Fig. 6.5.13

Step 5 : Logic diagram

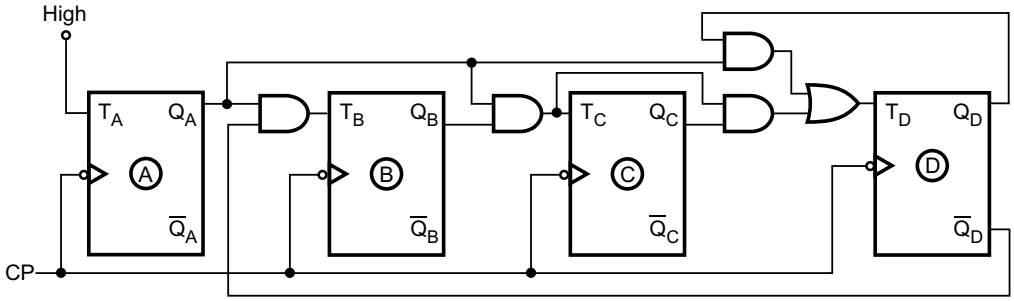


Fig. 6.5.14

Step 6 : Timing diagram

Fig. 6.5.15 shows the timing diagram for the synchronous decade counter.

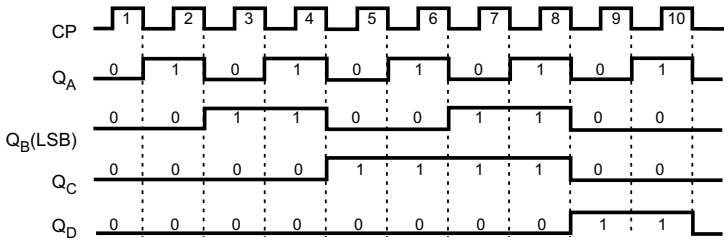


Fig. 6.5.15 Timing diagram

Example 6.5.9 *Design a 3-bit synchronous updown counter using T Flip-Flops.*

Solution : Table 6.5.12 shows the excitation table for 3-bit up/down synchronous counter using T flip-flops.

Excitation table

Input	Present State			Next State			Flip-flop Inputs		
UP/$\overline{\text{DOWN}}$ (UD)	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	T_C	T_B	T_A
0	0	0	0	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0	1	1
0	0	1	1	0	1	0	0	0	1
0	1	0	0	0	1	1	1	1	1
0	1	0	1	1	0	0	0	0	1
0	1	1	0	1	0	1	0	1	1
0	1	1	1	1	1	0	0	0	1
1	0	0	0	0	0	1	0	0	1
1	0	0	1	0	1	0	0	1	1
1	0	1	0	0	1	1	0	0	1
1	0	1	1	1	0	0	1	1	1
1	1	0	0	1	0	1	0	0	1
1	1	0	1	1	1	0	0	1	1
1	1	1	0	1	1	1	0	0	1
1	1	1	1	0	0	0	1	1	1

Table 6.5.12

K-map simplification

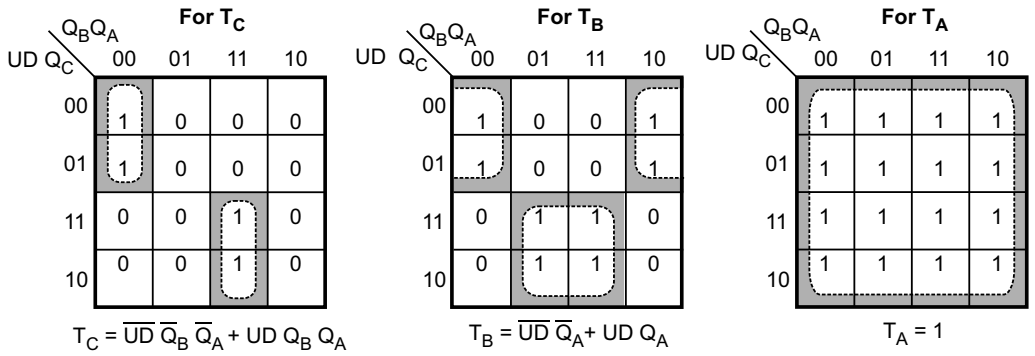


Fig. 6.5.16

Logic Diagram

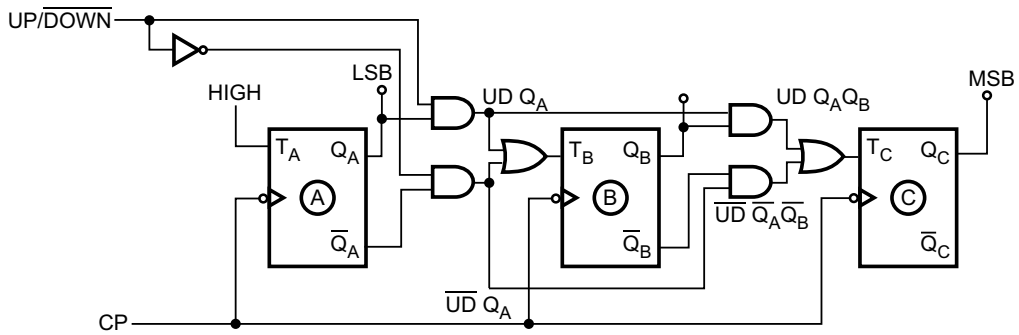


Fig. 6.5.17

Example 6.5.10 *Design a three bit binary counter using T flip-flops.*

Dec.-09, Dec.-11, Marks 16

Solution : Table 6.5.13 shows the excitation table for 3-bit binary counter

Present state			Next state			Flip-flop inputs		
A	B	C	A_+	B_+	C_+	T_A	T_B	T_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Table 6.5.13

K-map simplification

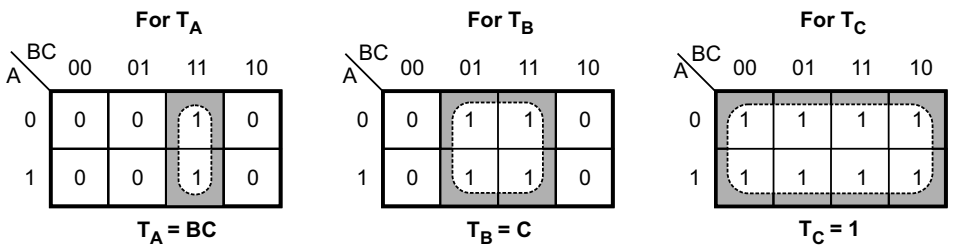


Fig. 6.5.18

Logic diagram

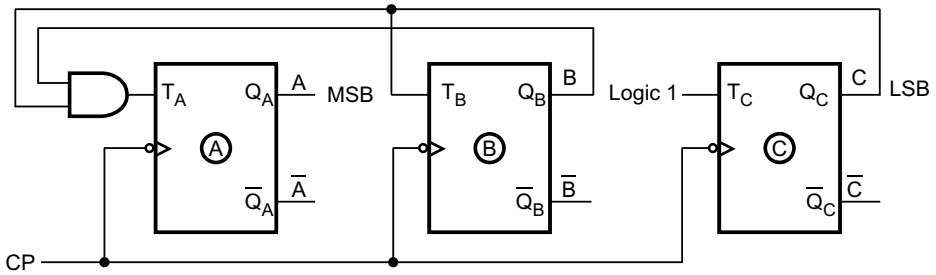


Fig. 6.5.19

Example 6.5.11 *Design and explain the working of a synchronous mod-3 counter.*

May-03, Marks 16

Solution : N = 3 and

Step 1 : Since $2^2 > 3$, n = 2 i.e. Flip-Flops needed = 2.

Step 2 : Flip-Flops used : JK

Step 3 : Transition table

Present state		Next state		Flip-flop inputs			
Q_B	Q_A	Q_{B+1}	Q_{A+1}	J_B	K_B	J_A	K_A
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	0	0	X	1	0	X
1	1	X	X	X	X	X	X

Table 6.5.14

Note : Refer section 5.5.3.2 for excitation table.

Step 4 : K-map simplification

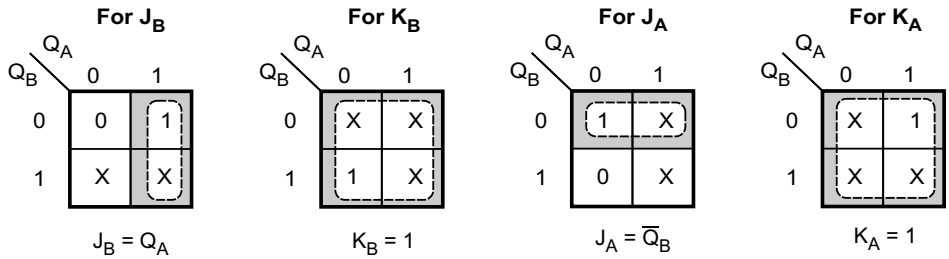


Fig. 6.5.20 (a)

Step 5 : Logic diagram

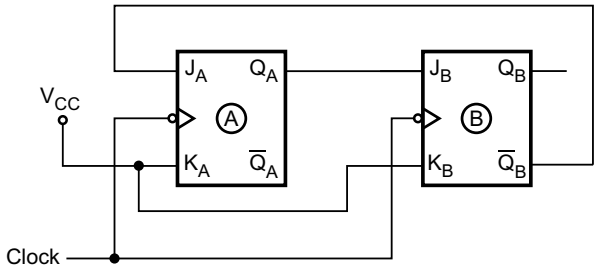


Fig. 6.5.20 (b)

Example 6.5.12 *Design and explain the working of mod-7 counter.* **Dec.-03, Marks 8**

Solution :

Step 1 : N = 7, and since $2^3 > 7$, n = 3 i.e. Flip-Flops needed = 3

Step 2 : Flip-Flops used : JK

Step 3 : Transition table

Present state			Next state			Flip-flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	0	X	0	X	1	X
1	0	1	1	1	0	0	X	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	X	X	X	X	X	X	X	X	X

Table 6.5.15

Note : Refer section 5.5.3.2 for excitation table.

Step 4 : K-map simplification

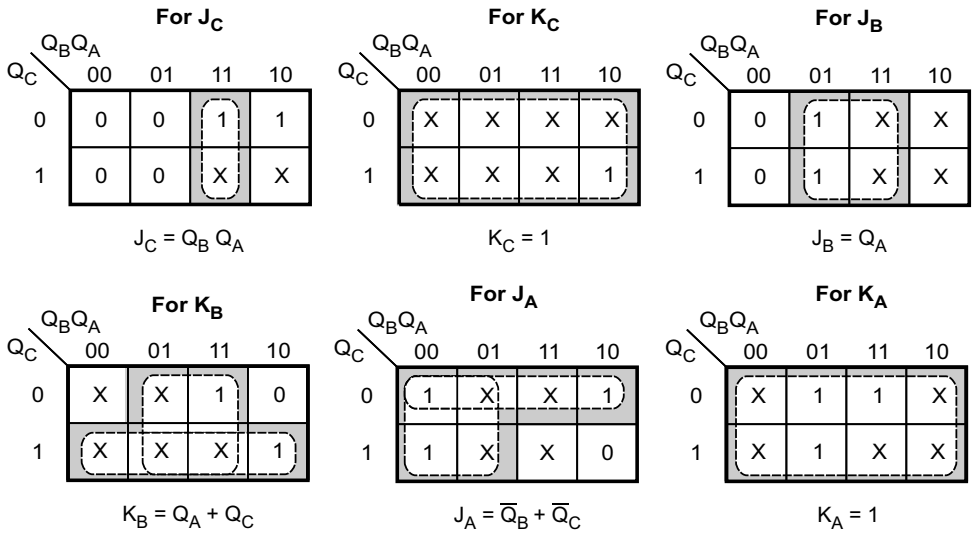


Fig. 6.5.21 (a)

Step 5 : Logic diagram

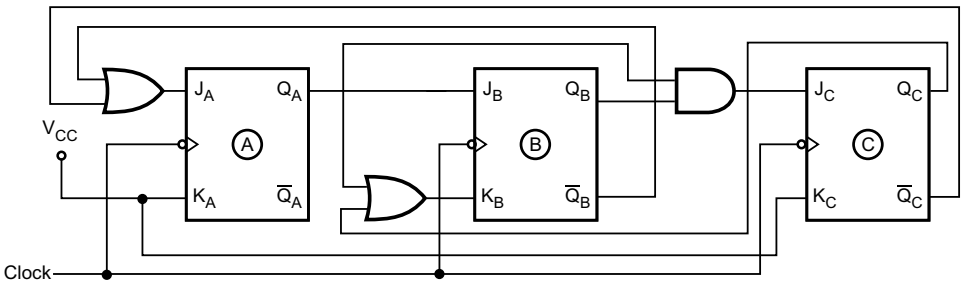


Fig. 6.5.21 (b)

Example 6.5.13 *Design a synchronous counter with states 0, 1, 2, 3, 0, 1. ..... using JK FFs.* **May-04, Marks 16**

Solution :

Step 1 : Here, N = 4 and since $2^2 \geq 4$ we need 2 Flip-Flops

Step 2 : Flip-Flops to be used : JK

Step 3 : Transition table

Present state		Next state		Flip-Flop inputs			
Q_B	Q_A	Q_{B+1}	Q_{A+1}	J_B	K_B	J_A	K_A
0	0	0	1	0	X	1	X
0	1	1	0	1	X	X	1
1	0	1	1	X	0	1	X
1	1	0	0	X	1	X	1

Table 6.5.16

Note : Refer section 5.5.3.2 for excitation table.

Step 4 : K-map simplification

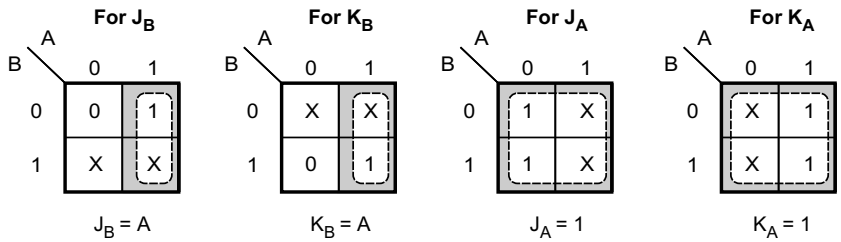


Fig. 6.5.22 (a)

Step 5 : Logic diagram

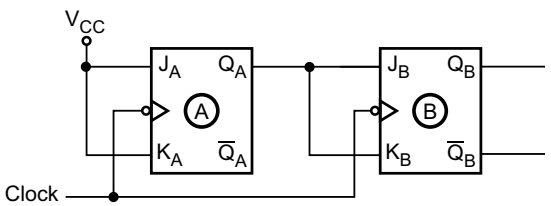


Fig. 6.5.22 (b)

Example 6.5.14 *Design a 3-bit binary counter using T flip-flop that has a repeated sequence of six states.* 000 - 001 - 010 - 100 - 101 - 110. *Give the state table, state diagram and logic diagram.* **Dec.-07, Marks 16**

Solution : Step 1 : State diagram

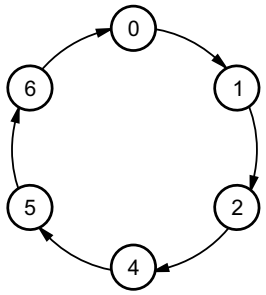


Fig. 6.5.23 (a)

Step 2 : State table

Present state			Next state			Flip-flop inputs		
A	**B**	**C**	A^+	B^+	C^+	T_A	T_B	T_C
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	1	0	0	1	1	0
0	1	1	0	0	0	0	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	0	0	0	1	1	0
1	1	1	0	0	0	1	1	1

Upon power on, if counter is in unused states, it is reset to 000.

Step 3 : K-map simplification

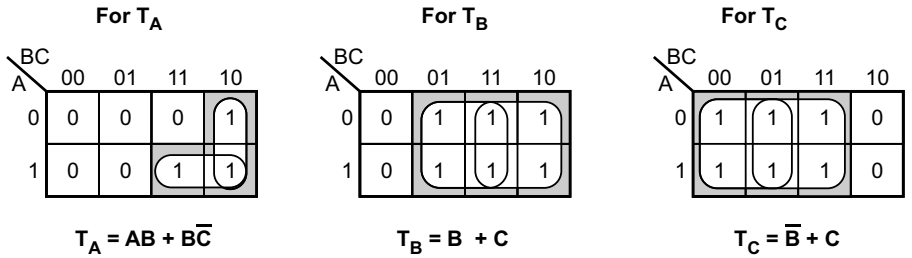


Fig. 6.5.23 (b)

Step 4 : Logic diagram

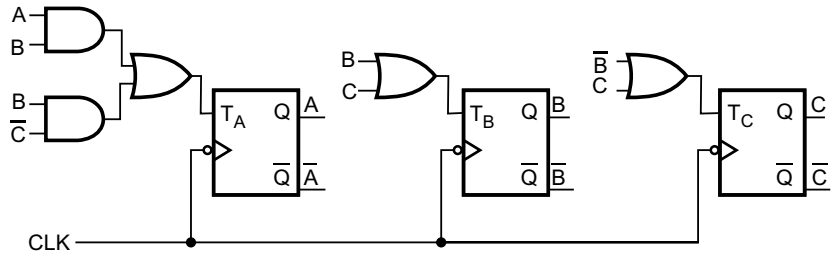


Fig. 6.5.23 (c)

Example 6.5.15 *Design a synchronous up/down counter that will count up from zero to one to two to three and will repeat whenever an external input x is logic 0, and will count down from three to two to one to zero and will repeat whenever the external input x is logic 1. Implement your circuit with one TTL SN74LS76 device and one TTL SN74LS00 device.* **May-10, Marks 12**

Solution : Step 1 : Excitation table

Input	Present state		Next state		Flip-Flop Inputs			
X	A	B	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	1	0	X	1	X
0	0	1	1	0	1	X	X	1
0	1	0	1	1	X	0	1	X
0	1	1	0	0	X	1	X	1
1	0	0	1	1	1	X	1	X
1	0	1	0	0	0	X	X	1
1	1	0	0	1	X	1	1	X
1	1	1	1	0	X	0	X	1

Step 2 : K-map simplification

$\therefore \quad J_A = K_A = \overline{X}B + X\overline{B}$

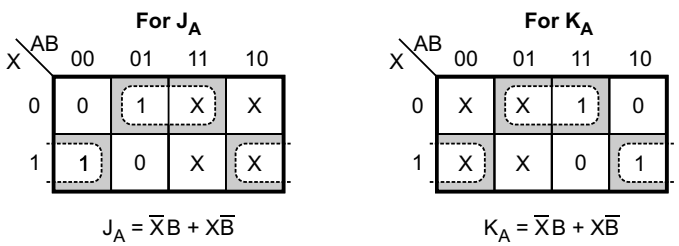


Fig. 6.5.24

Step 3 : Logic diagram

We can implement combinational logic circuit for J_A and K_A input using NAND-NAND logic, as shown in the Fig. 6.5.24.

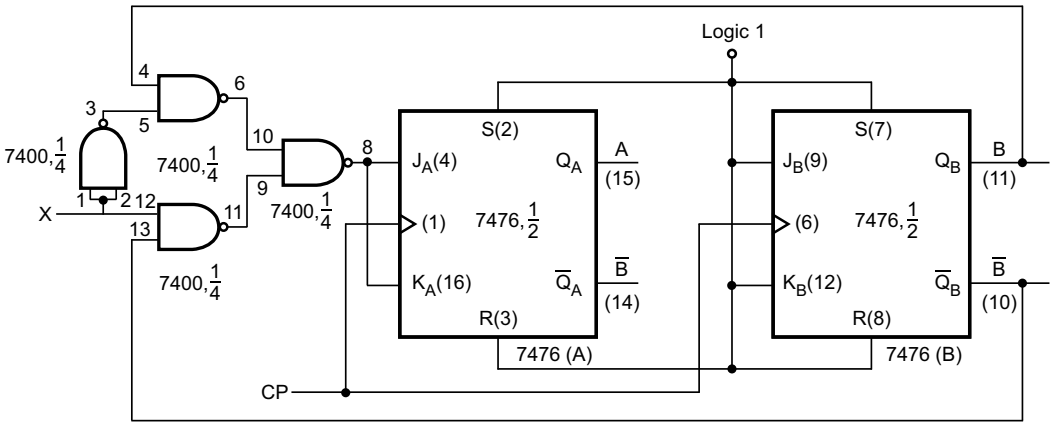


Fig. 6.5.24 (a)

Examples for Practice

Example 6.5.16 *Design a 3 bit synchronous gray code counter using T flip-flop.*

Example 6.5.17 *The following sequence is to be realized by a counter consisting of 3 JK FF's.*

A_1	0	0	0	0	1	1	0
A_2	0	1	1	0	0	1	0
A_3	0	1	0	1	1	0	0

Design the counter.

Example 6.5.18 *Design and explain the working of a mod-11 counter.* **May-05, Marks 8**

Example 6.5.19 *Design and draw the output waveform of UP/DOWN counter using JK-FF.* **May-12, Marks 16**

Review Question

1. *Explain the design steps of Mod n counter.* **May-12, Marks 8**

6.6 Two Marks Questions with Answers

Q.1 *What is the minimum number of flip-flops needed to design a counter of modulus 60 ?* **May-04**

Ans. : $2^n \geq 60 \quad \therefore \quad n = 6$

Q.2 *What is the minimum number of flip-flops required to implement a modulo 21 synchronous counter ?* **Dec.-03**

Ans. : $2^n \geq 21 \quad \therefore \quad n = 5$

Q.3 *What is meant by programmable counter ? Mention its application.* **May-10**

Ans. : A counter that divides an input frequency by a number which can be programmed, is called **programmable counter**.

Applications of programmable counter

1. Frequency division
2. Digital clock
3. Stop watch
4. Programmable logic controllers.

Q.4 *Name two sequential switching circuits.* **May-03**

Ans. : Counter, shift register.

Q.5 *Define synchronous counter.* **Dec.-06**

Ans. : When counter is clocked such that each flip-flop in the counter is triggered at the same time, the counter is called synchronous counter.

Q.6 *Define asynchronous counter.*

Ans. : The counter in which the output of the current Flip-Flop drives the clock input of the next higher-order Flip-Flop is called asynchronous counter.

Q.7 *Draw the state diagram of MOD-10 counter.* **Dec.-08**

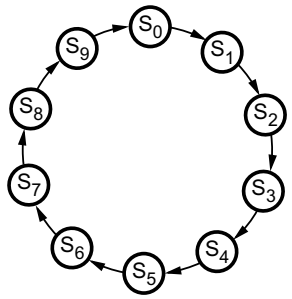


Fig. 6.6.1

❑❑❑

7 Shift Registers

Syllabus

Registers - Shift registers - Universal shift registers - Shift register counters - Ring counter - Shift counters - Sequence generators.

Contents

7.1 Introduction

7.2 Shift Registers

7.3 Types of Shift Registers ***May-06,***

. . . . . ***Dec.-06, 12,*** · · · · · · · · · · · · *Marks 8*

7.4 Universal Shift Register ***May-09, 10,***

. . . . . ***Dec.-08, 10,*** · · · · · · · · · · · *Marks 12*

7.5 Applications of Shift Registers

7.6 Ring Counters . . . . . ***Dec.-04, 05, 12, May-12,*** · *Marks 16*

7.7 Johnson or Twisting Ring or Switch Tail Counter . . ***May-07,***

. . . . . ***Dec.-06, 11, 12,*** · · · · · · · · · *Marks 8*

7.8 Two Marks Questions with Answers

7.1 Introduction

We have seen that a flip-flop is nothing but a binary cell capable of storing one bit information, and can be connected together to perform counting operations. Such a group of flip-flops is called **counter**. We have also seen that group of flip-flops can be used to store a word, which is called **register**.

A flip-flop can store 1-bit information. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

7.1.1 Buffer Register

Fig. 7.1.1 shows the simplest register constructed with four D flip-flops. This register is also called **buffer register**. Each D flip-flop is triggered with a common negative edge clock pulse. The input bits set up the flip-flops for loading. Therefore, when the first negative clock edge arrives, the stored binary information becomes,

$Q_A Q_B Q_C Q_D = ABCD$

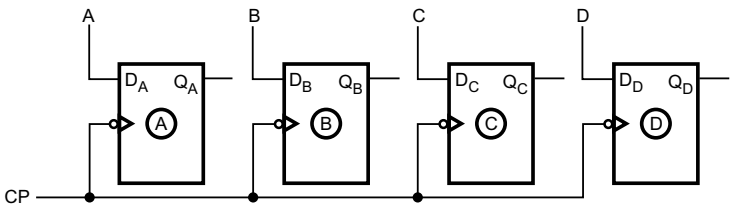


Fig. 7.1.1 Buffer register

In this register, four D flip-flops are used. So it can store 4-bit binary information. Thus the number of flip-flop stages in a register determines its total storage capacity.

7.1.2 Controlled Buffer Register

We can control input and output of the register by connecting tri-state devices at the input and output sides of register as shown in Fig. 7.1.2. So this register is called '**controlled buffer register**'.

Here, tri-state switches are used to control the operation. When you want to store data in the register, you have to make $\overline{LOAD}$ or $\overline{WR}$ signal low to activate the tri-state buffers. When you want the data at the output, you have to make $\overline{RD}$ signal low to activate the buffers. Controlled buffer registers are commonly used for temporary storage of data within a digital system.

As seen above the 4-bit register can store 4-bit binary information. In general, n-bit register can store n-bit binary information.

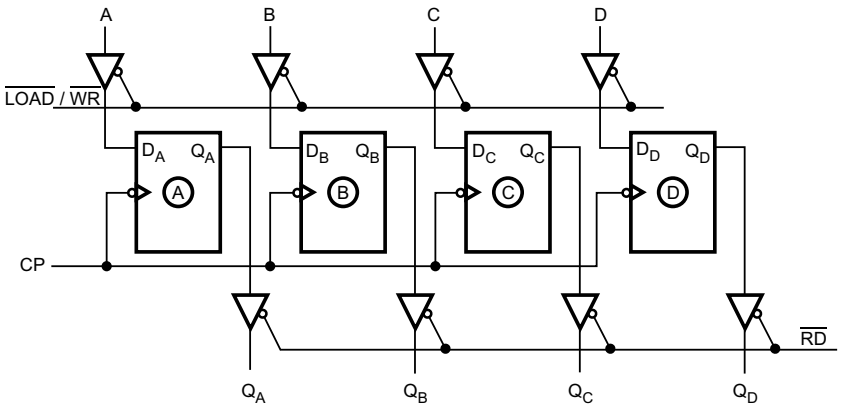


Fig. 7.1.2 Controlled buffer register

Example 7.1.1 *Determine the number of flip-flops needed to construct a register capable of storing,*

i) A 6-bit binary number *ii) Decimal numbers up to 32*

iii) Hexadecimal numbers up to F *iv) Octal numbers up to 10.*

Solution :

i) A 6-bit binary number requires register with 6 flip-flops.

ii) $(32)_{10} = (100000)_2$. The number of bits required to represent 32 in binary are six, therefore, 6 flip-flops are needed to construct a register capable of storing 32 decimal.

iii) $(F)_{16} = (1111)_2$. The number of bits required to represent $(F)_{16}$ in binary are four, therefore four flip-flops are needed to construct a register capable of storing $(F)_{16}$.

iv) $(10)_8 = (1000)_2$. The number of bits required to represent $(10)_8$ in binary are four, therefore, four flip-flops are needed to construct a register capable of storing $(10)_8$.

Review Questions

1. *What is register ?*
2. *What is buffer register ?*
3. *What do you mean by controlled buffer register ?*

7.2 Shift Registers

The binary information (data) in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to a group of registers called '**shift registers**'. They are very important in applications involving the storage and transfer of data in a digital system.

Fig. 7.2.1 gives the symbolical representation of the different types of data movement in shift register operations.

According to the data movement in a register, let us see some of the types of shift registers.

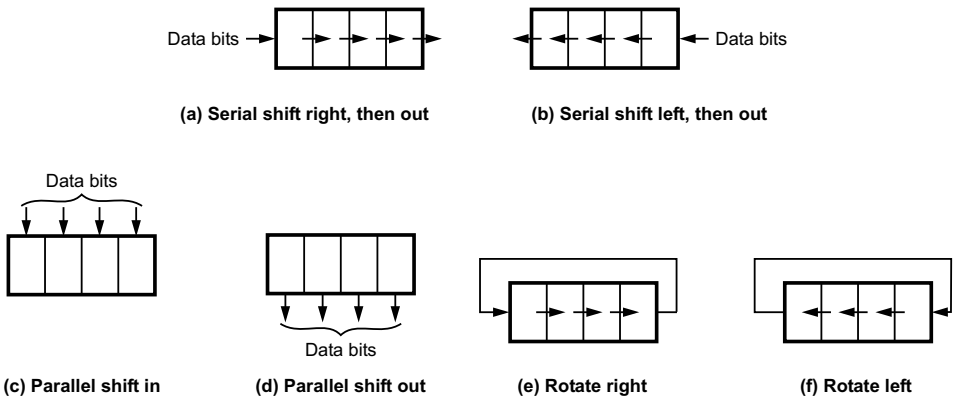


Fig. 7.2.1 Basic data movement in registers

Review Question

1. *List the basic types of shift registers in terms of data movement.*

7.3 Types of Shift Registers

May-06, Dec.-06,12

7.3.1 Serial In Serial Out (SISO) Shift Register

Shift Left Mode

Fig. 7.3.1 shows serial-in serial-out shift-left register.

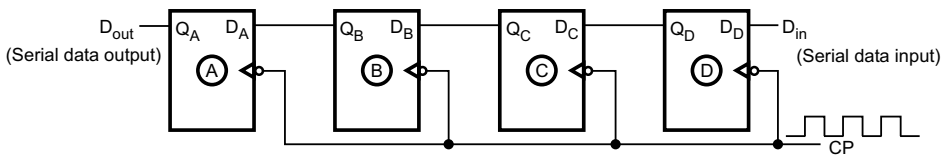
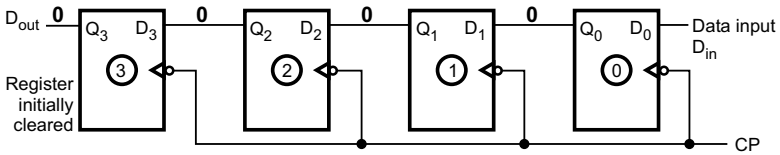


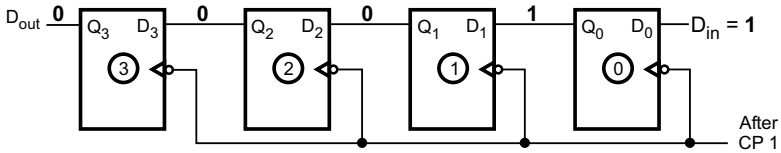
Fig. 7.3.1 Shift-left register

We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit. Initially, register is cleared. So $Q_A Q_B Q_C Q_0$ = 0 0 0 0



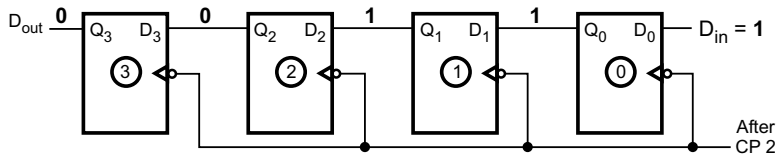
(a)

a) When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as D_{in}, D_{in} = 1, $Q_3Q_2Q_1Q_0$ = 0 0 0 0. The arrival of the first falling clock edge sets the right-most flip-flop, and the stored word becomes, $Q_3Q_2Q_1Q_0$ = 0 0 0 1



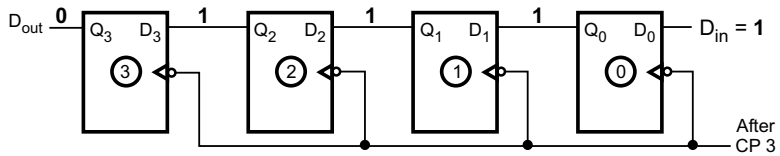
(b)

b) When the next negative clock edge hits, the Q_1 flip-flop sets and the register contents become, $Q_3Q_2Q_1Q_0$ = 0 0 1 1



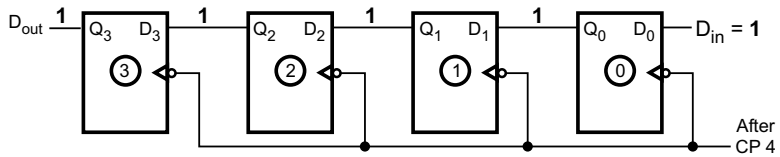
(c)

c) The third negative clock edge results in, $Q_3Q_2Q_1Q_0$ = 0 1 1 1



(d)

d) The fourth falling clock edge gives, $Q_3Q_2Q_1Q_0$ = 1 1 1 1.



(e)

Fig. 7.3.2 Four bits 1111 being serially entered into shift-left register

The Table 7.3.1 summarizes the shift left operation.

CP	Q_3	Q_2	Q_1	Q_0	D_{in}
Initially	0	0	0	0	1
↓ 1st	0	0	0	1	1
↓ 2nd	0	0	1	1	1
↓ 3rd	0	1	1	1	1
↓ 4th	1	1	1	1	1

Table 7.3.1 Shift left operation

The Fig. 7.3.3 shows waveforms for shift left operation

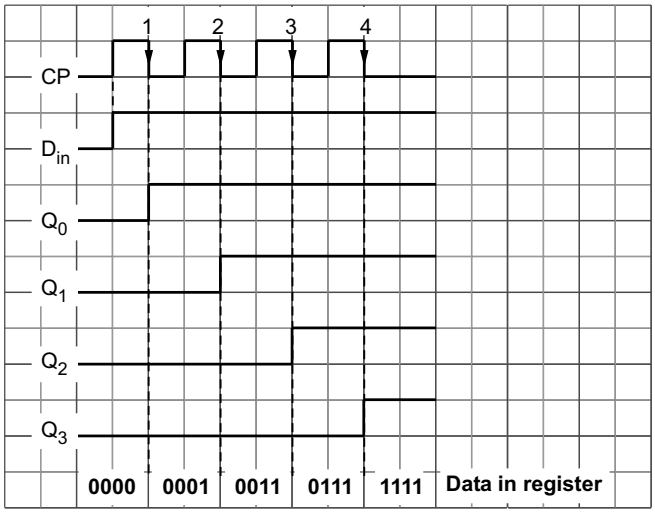


Fig. 7.3.3 Waveforms for shift left register

Shift Right Mode

Fig. 7.3.4 shows serial-in serial-out shift-left register.

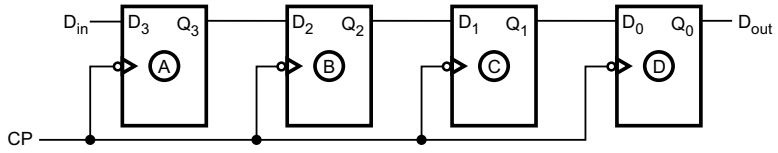
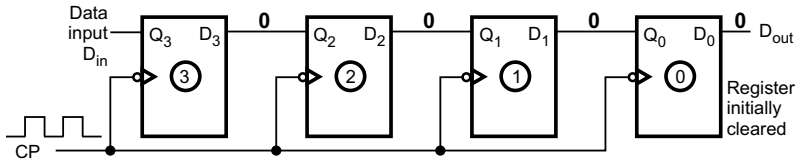


Fig. 7.3.4 Shift-right register

We will illustrate the entry of the four bit binary number 1111 into the register, beginning with the left-most bit.

Initially, register is cleared. So $Q_3Q_2Q_1Q_0$ = 0 0 0 0

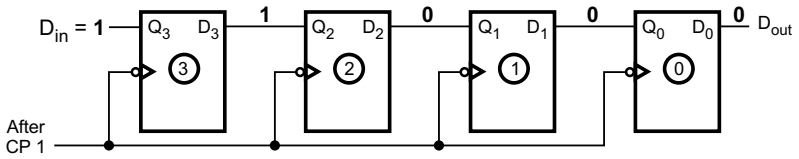


(a)

a) When data 1 1 1 1 is applied serially, i.e. left-most 1 is applied as D_{in},

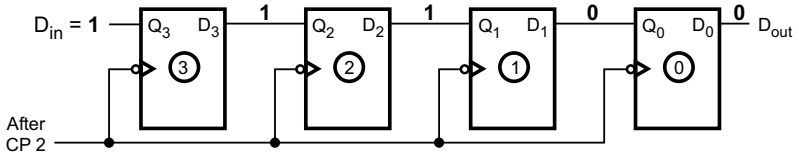
D_{in} = 1, $Q_3Q_2Q_1Q_0$ = 0 0 0 0

The arrival of the first falling clock edge sets the left-most flip-flop, and the stored word becomes, $Q_3Q_2Q_1Q_0$ = 1 0 0 0



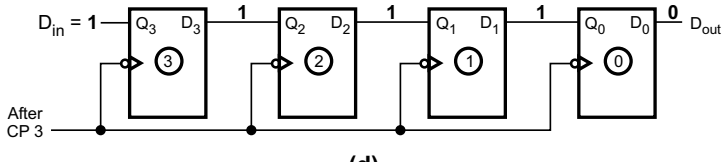
(b)

b) When the next falling clock edge hits, the Q_2 flip-flop sets and the register contents become, $Q_3Q_2Q_1Q_0$ = 1100



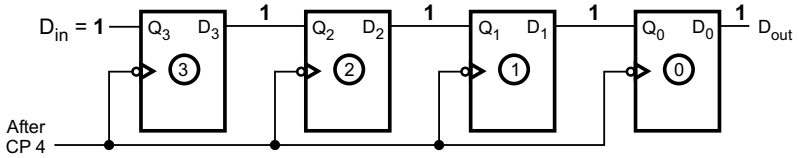
(c)

c) The third falling clock edge results in, $Q_3Q_2Q_1Q_0$ = 1110.



(d)

d) The fourth falling clock edge gives, $Q_3Q_2Q_1Q_0$ = 1111.



(e)

Fig. 7.3.5 Four bits 1111 being serially entered into shift-right register

Table 7.3.2 summarizes the shift right operation.

CP	D_{in}	Q_3	Q_2	Q_1	Q_0
Initially	1	0	0	0	0
↓ 1st	1	1	0	0	0
↓ 2nd	1	1	1	0	0
↓ 3rd	1	1	1	1	0
↓ 4th	1	1	1	1	1

Table 7.3.2 Shift right operation

The Fig. 7.3.6 shows waveforms for shift right operation.

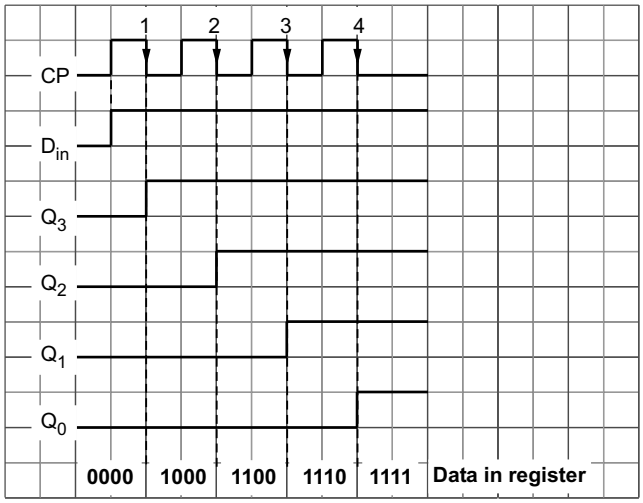


Fig. 7.3.6 Waveforms for shift right register

7.3.2 Serial In Parallel Out (SIPO) Shift Register

In this case, the data bits are entered into the register in the same manner as discussed in the last section, i.e. serially. But the output is taken in parallel. Once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, instead of on a bit-by-bit basis as with the serial output as shown in Fig. 7.3.7.

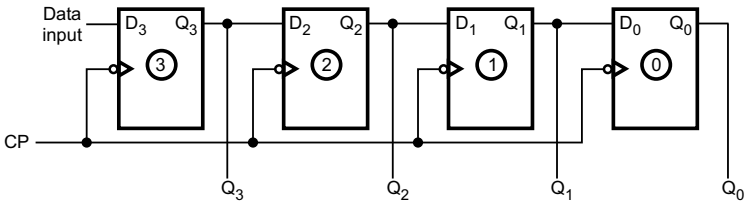


Fig. 7.3.7 A Serial In Parallel Out (SISO) shift register

CP	Q_3	Q_2	Q_1	Q_0
–	NC	NC	NC	NC
↓	D_3	D_2	D_1	D_0

Table 7.3.3 Truth table

7.3.3 Parallel In Serial Out (PISO) Shift Register

In this type, the bits are entered in parallel i.e simultaneously into their respective stages on parallel lines.

Fig. 7.3.8 illustrates a four-bit parallel in serial out register. There are four input lines A_3, A_2, A_1, A_0 for entering data in parallel into the register. SHIFT/$\overline{\text{LOAD}}$ is the control input which allows shift or loading data operation of the register. When SHIFT/$\overline{\text{LOAD}}$ is low, gates G_1, G_2, G_3 are enabled, allowing each input data bit to be applied to D input of its respective flip-flop. When a clock pulse is applied, the flip-flops with D = 1 will SET and those with D = 0 will RESET. Thus all four bits are stored simultaneously.

When SHIFT/$\overline{\text{LOAD}}$ is high, gates G_1, G_2, G_3 are disabled and gates G_4, G_5, G_6 are enabled. This allows the data bits to shift right from one stage to the next. The OR gates at the D-inputs of the flip-flops allow either the parallel data entry operation or shift operation, depending on which AND gates are enabled by the level on the SHIFT/$\overline{\text{LOAD}}$ input.

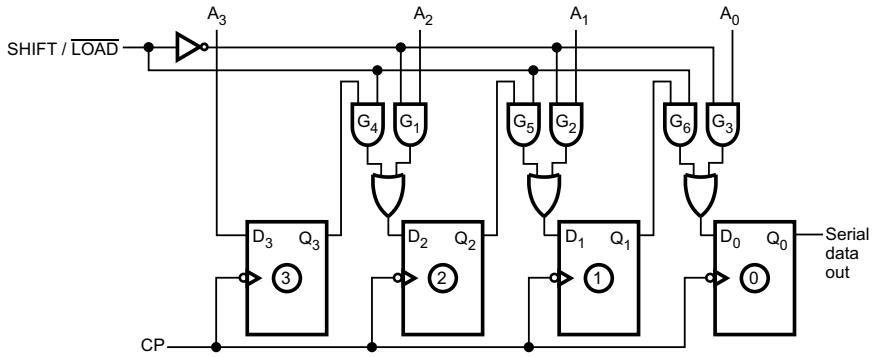


Fig. 7.3.8 Parallel In Serial Out (PISO) shift register

7.3.4 Parallel In Parallel Out (PIPO) Shift Register

From the third and second types of registers, it is cleared that how to enter the data in parallel i.e. all bits simultaneously into the register and how to take data out in parallel from the register. In 'parallel in parallel out register', there is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. Fig. 7.3.9 shows this type of register.

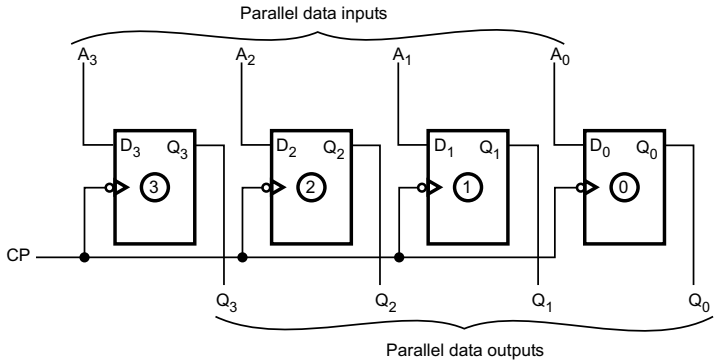


Fig. 7.3.9 Parallel In Parallel Out (PIPO) shift register

7.3.5 Bidirectional Shift Register

This type of register allows shifting of data either to the left or to the right side. It can be implemented by using logic gate circuitry that enables the transfer of data from one stage to the next stage to the right or to the left, depending on the level of a control line. Fig. 7.3.10 illustrates a four-bit bidirectional register. The RIGHT/$\overline{\text{LEFT}}$ is the control input signal which allows data shifting either towards right or towards left. A high on this line enables the shifting of data towards right and a low enables it towards left. When RIGHT/$\overline{\text{LEFT}}$ signal is high, gates G_1, G_2, G_3, G_4 are enabled. The state of the Q output of each flip-flop is passed through the D input of the following flip-flop. When a clock pulse arrives, the data are shifted one place to the right. When the RIGHT/$\overline{\text{LEFT}}$ signal is low, gates G_5, G_6, G_7, G_8 are enabled. The Q output of each flip-flop is passed through the D input of the preceding flip-flop. When clock pulse arrives, the data are shifted one place to the left.

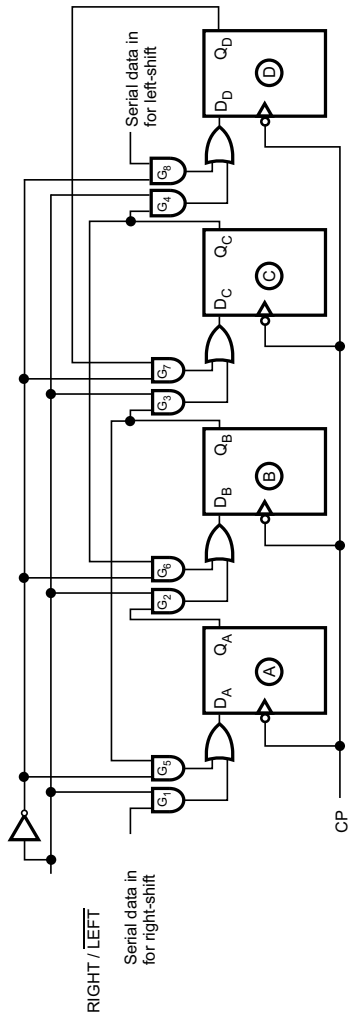


Fig. 7.3.10 4-bit bidirectional shift register

Bidirectional Shift Register with Parallel Load

We have seen that shift register can be used for converting serial data into parallel data, and vice versa. When parallel load capability is added to the shift register, the data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Such a register is called bidirectional shift register with parallel load. Fig. 7.3.11 shows bidirectional shift register with parallel load.

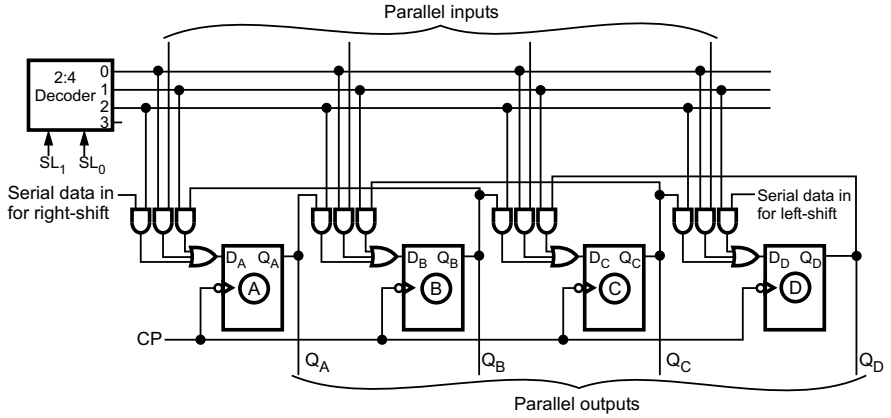


Fig. 7.3.11 4-bit bidirectional shift register with parallel load

As shown in the Fig. 7.3.11, the D input of each flip-flop has three sources : Output of left adjacent flip-flop, output of right adjacent flip-flop and parallel input. Out of these three sources one source is selected at a time and it is done with the help of decoder. The decoder select lines (SL_1 and SL_0) select the one source out of three as shown in the Table 7.3.4.

SL_1	SL_0	Selected Source
0	0	Parallel input
0	1	Output of right adjacent FF
1	0	Output of left adjacent FF
1	1	–

Table 7.3.4

When select lines are 00 (i.e. $SL_1 = 0$ and $SL_0 = 0$), data from the parallel inputs is loaded into the 4-bit register. When select lines are 01 (i.e. $SL_1 = 0$ and $SL_0 = 1$), data within the register is shifted 1-bit left. When select lines are 10 (i.e. $SL_1 = 1$ and $SL_0 = 0$), data within the register is shifted 1-bit right.

Review Questions

1. *Draw a 4-bit serial-in-serial-out shift register and draw its waveforms.*

May-06, Dec.-06, Marks 8

2. *Draw a 4-bit parallel-in-serial-out shift register and briefly explain.* **May-06, Marks 8**
3. *Draw the 8-bit serial-in-parallel-out shift register and explain its operation.* **Dec.-06, Marks 8**
4. *Draw and explain the operation of parallel-in-parallel-out shift register.*
5. *Explain the operation of 4-bit bidirectional shift register with the help of neat diagram.*
6. *Design a 4 bit bi-directional shift register.* **Dec.-12, Marks 8**

7.4 Universal Shift Register

May-09, 10, Dec.-08, 10

A register capable of shifting in one direction only is a unidirectional shift register. A register capable of shifting in both directions is a bidirectional shift register. If the register has both shifts (right shift and left shift) and parallel load capabilities, it is referred to as **Universal shift register**.

The Fig. 7.4.1 shows the 4-bit universal shift register. It has all the capabilities listed above. It consists of four flip-flops and four multiplexers. The four multiplexers have two common selection inputs S_1 and S_0, and they select appropriate input for D flip-flop. The Table 7.4.1 shows the register operation depending on the selection inputs of multiplexers. When S_1S_0 = 00, input 0 is selected and the present value of the

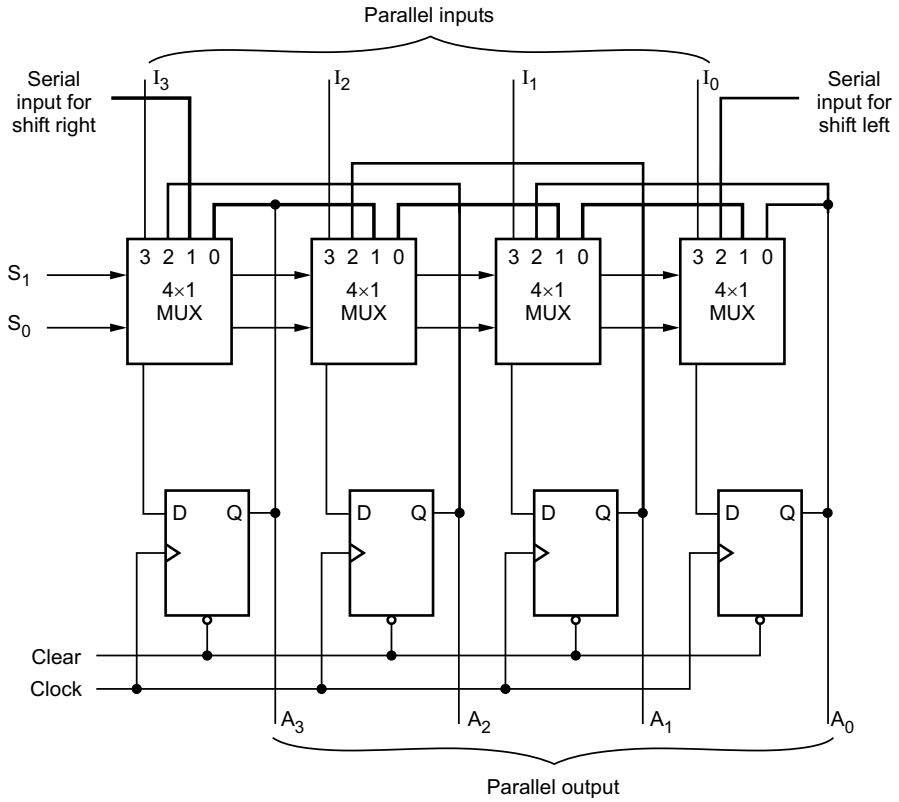


Fig. 7.4.1 4-bit universal shift register

register is applied to the D inputs of the flip-flops. This results no change in the register value. When S_1S_0 = 01, input 1 is selected and circuit connections are such that it operates as a right shift register. When S_1S_0 = 10, input 2 is selected and circuit connections are such that it operates as a left shift register. Finally, when S_1S_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously and it is a parallel load operation.

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Table 7.4.1 Mode control and register operation

Review Questions

Q.1 *What is meant by universal shift register? Explain the principle of operation of 4-bit universal shift register.* **May-10, Marks 12**

Q.2 *Explain the operation of universal shift register with neat block diagram.* **Dec.-08, 10, May-09, Marks 10**

7.5 Applications of Shift Registers

We have seen that primary use of shift register is temporary data storage and bit manipulations. Some of the common applications of shift registers are as discussed below.

7.5.1 Delay Line

A Serial-In-Serial-Out (SISO) shift register can be used to introduce time delay Δt in digital signals. The time delay can be given as

$$\Delta t = N \times \frac{1}{f_c}$$

where N is the number of stages (i.e. flip-flops) and f_c is the clock frequency.

Thus, an input pulse train appears at the output delayed by Δt. The amount of delay can be controlled by the clock frequency or by the number of flip-flops in the shift register.

7.5.2 Serial-to-Parallel Converter

A Serial-In-Parallel-Out (SIPO) shift register can be used to convert data in the serial form to the parallel form.

7.5.3 Parallel-to-Serial Converter

A Parallel-In-Serial-Out (PISO) shift register can be used to convert data in the parallel form to the serial form.

7.5.4 Shift Register Counters

A shift register can also be used as a counter. A shift register with the serial output connected back to the serial input is called **shift register counter**. Because of such a connection, special specified sequences are produced as the output. The most common shift register counters are the ring counter and the Johnson counter.

7.5.5 Pseudo-Random Binary Sequence (PRBS) Generator

Another important application of shift register is a pseudo-random binary sequence generator. Here, suitable feedback is used to generate pseudo-random sequence. The term random here means that the outputs do not cycle through a normal binary count sequence. The term pseudo here refers to the fact that the sequence is not truly random because it does cycle through all possible combinations once every $2^n - 1$ clock cycles, where n represents the number of shift register stages (number of flip-flops).

7.5.6 Sequence Generator

The shift register can be used to generate a particular bit pattern repetitively. The Fig. 7.5.1 shows the basic block diagram of a sequence generator. Here, left most flip-flop input accept the serial input and the right most flip-flop gives serial data output. It is important to note that the serial data output signal is connected as a serial data in. On every clock pulse the data shift operation takes place. We get the loaded bit pattern at the serial output in a sequence. Same bit pattern is again loaded in the register since serial output is connected serial in of the register. Thus, the circuit generates a particular bit pattern repetitively.

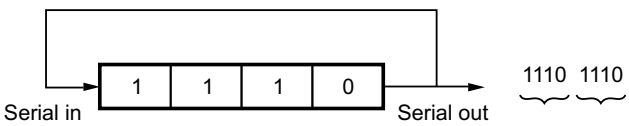


Fig. 7.5.1 4-bit sequence generator

7.5.7 Sequence Detector

The shift register can be used to detect the desired sequence. The detection process requires two registers : one register stores the bit pattern to be detected i.e. R_1 and other

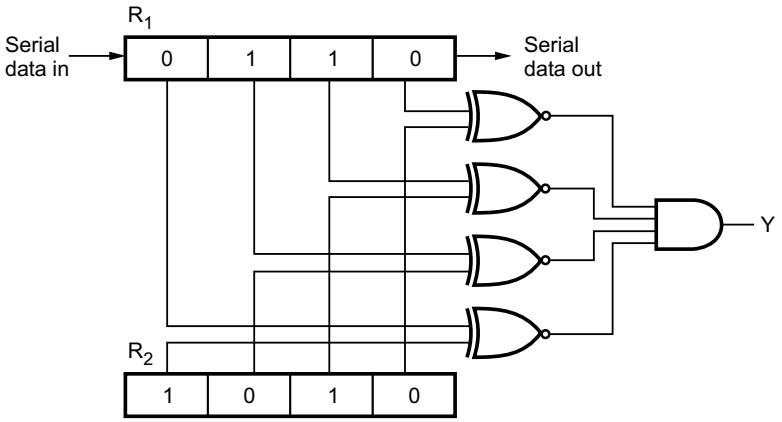


Fig. 7.5.2 4-bit sequence detector

register accepts the input data stream i.e. R_2. Input data stream enters a shift register as serial data in and leaves as serial out. In every clock cycle, bit-wise comparisons of these two registers are done using EX-NOR gates as shown in the Fig. 7.5.2. We know that, the two-input EX-NOR gate gives logic high output when both inputs are either low or high, i.e. when both are equal. When outputs of all the EX-NORs gates are logic high we can say that all bits are matched and hence the desired bit pattern is detected. The final output which indicates that the pattern is detected is taken from four-input AND gate.

The 4-bit sequence detector shown in Fig. 7.5.2 can be made programmable by loading the desired 4-bit data in the register R_2.

Review Question

1. *Explain the applications of shift registers.*

7.6 Ring Counters

Dec.-04, 05, 12, May-12

Fig. 7.6.1 shows the logic diagram for four-bit ring counter. As shown in the Fig. 7.6.1, the Q output of each stage is connected to the D input of the next stage and the output of last stage is fed back to the input of first stage. The $\overline{CLR}$ followed by $\overline{PRE}$ makes the output of first stage to '1' and remaining outputs are zero, i.e. Q_A is one and Q_B, Q_C, Q_D are zero.

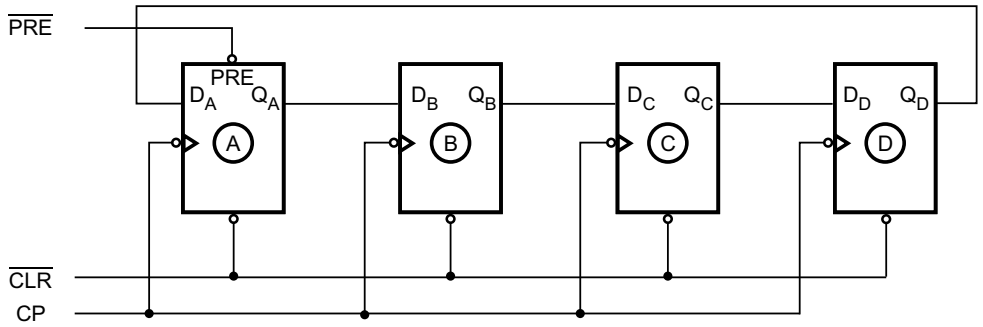


Fig. 7.6.1 Four-bit ring counter

The first clock pulse produces Q_B = 1 and remaining outputs are zero. According to the clock pulses applied at the clock input CP, a sequence of four states is produced. These states are listed in Table 7.6.1.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Table 7.6.1 Ring counter sequence 4-bits

As shown in Table 7.6.1, 1 is always retained in the counter and simply shifted 'around the ring', advancing one stage for each clock pulse. In this case four stages of flip-flops are used. So a sequence of four states is produced and repeated. Fig. 7.6.2 gives the timing sequence for a four-bit ring counter.

The ring counter can be used for counting the number of pulses. The number of pulses counted is read by noting which flip-flop is in state 1. No decoding circuitry is required. Since there is one pulse at the output for each of the N clock pulses, this circuit is also referred to as a divide-by-N-counter or an N : 1 scalar. Ring counters can be instructed for any desired MOD number, that is MOD-N ring counter requires N flip-flops.

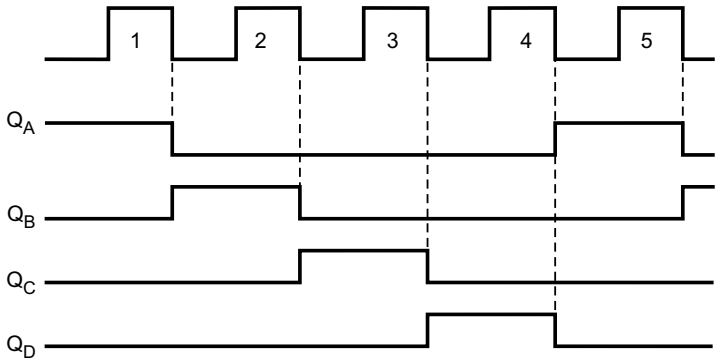


Fig. 7.6.2 Timing sequence for a four-bit ring counter

Self correcting counters

The ring counters suffer from one major problem-if its single 1 output is lost due to a temporary hardware problem (e.g. noise), the counter goes to state 0000 and stays there forever. Likewise, if an extra 1 output is set (i.e. state 0101 is created), the counter will go through an incorrect cycle of states and may stay in that cycle forever.

A self correcting counter is designed so that all abnormal states have transitions leading to normal states. The Fig. 7.6.3 (a) shows the 4-bit self correcting ring counter using IC 74X194. Here, the NOR gate is used to shift a 1 into D_{SL} only when the three least significant bits are 0. The Fig. 7.6.3 (b) shows how all abnormal states lead back into the normal cycle.

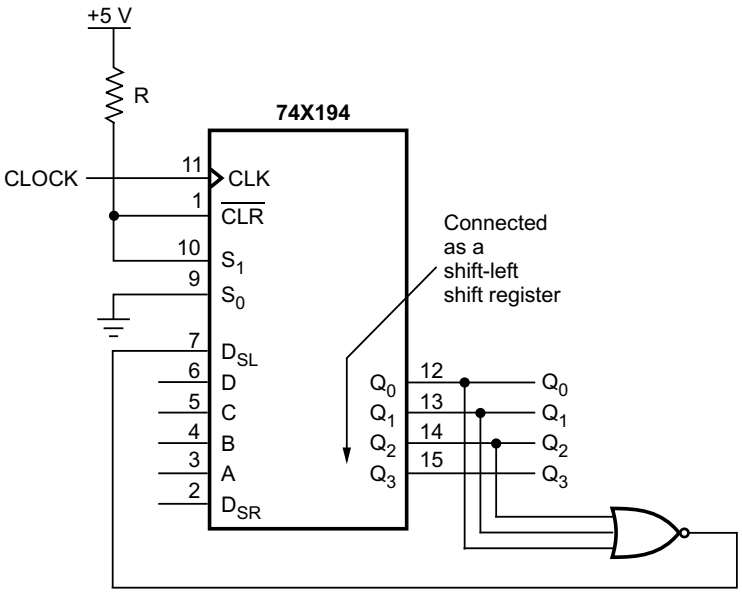


Fig. 7.6.3 (a) 4-bit self correcting ring counter

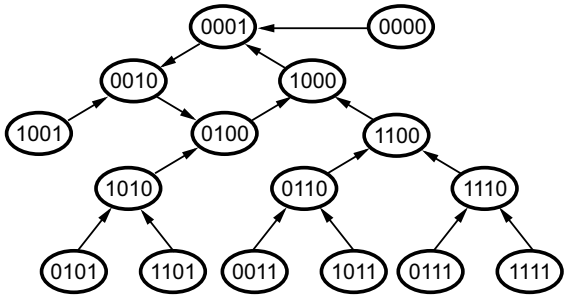


Fig. 7.6.3 (b) State diagram for self correcting ring counter

The ring counter shown above has a single circulating 1. The ring counter with a single circulating 0 can be designed using NAND gate instead of NOR gate in Fig. 7.6.3 (a).

Example 7.6.1 *Design a 4-bit, 4-state ring counter using 74X194.*

Solution : The Fig. 7.6.4 (a) shows the circuit diagram for a 4-bit, 4-state ring counter with a single circulating 1. Here, 74X194 universal shift register is connected so that it normally performs a left-shift. However, when RESET is asserted it loads 0001. Once RESET is negated, the 74194 shifts left on each clock pulse. The D_{SL} serial input is

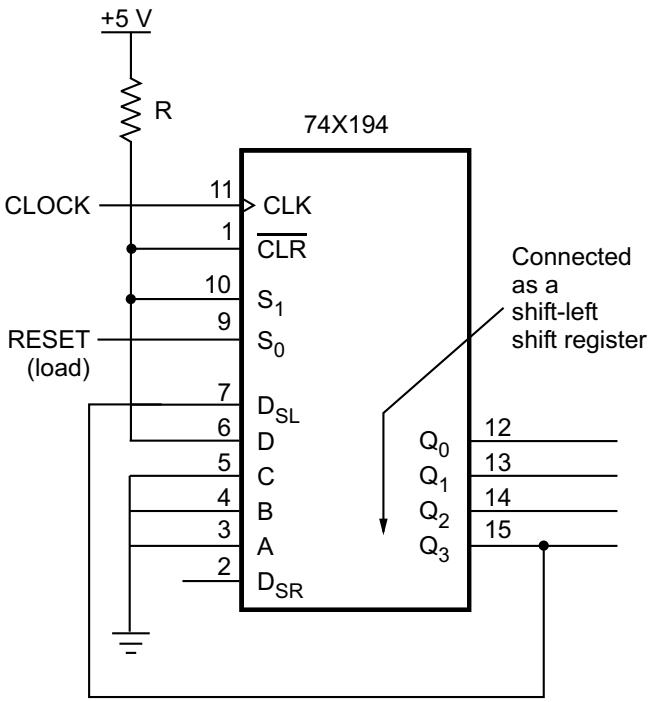


Fig. 7.6.4 (a) Circuit diagram

connected to the leftmost output (Q_3 : MSB), so the next states are 0010, 0100, 1000, 0001, 0010, ... Thus the counter visits four unique states before repeating. The Fig. 7.6.4 (b) shows the timing diagram for this 4-bit counter.

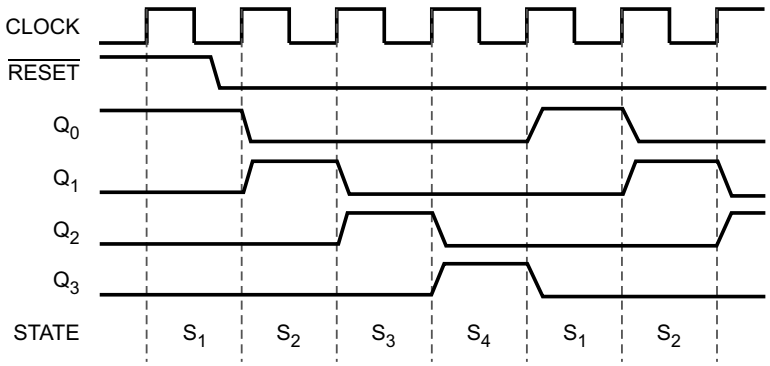


Fig. 7.6.4 (b) Timing diagram for a 4-bit ring counter

Example 7.6.2 *Assume that 1011 input data pattern is loaded into a 4-bit ring counter. Sketch the resulting flip-flop Q output waveforms (Assume positive edge triggering).*

Solution :

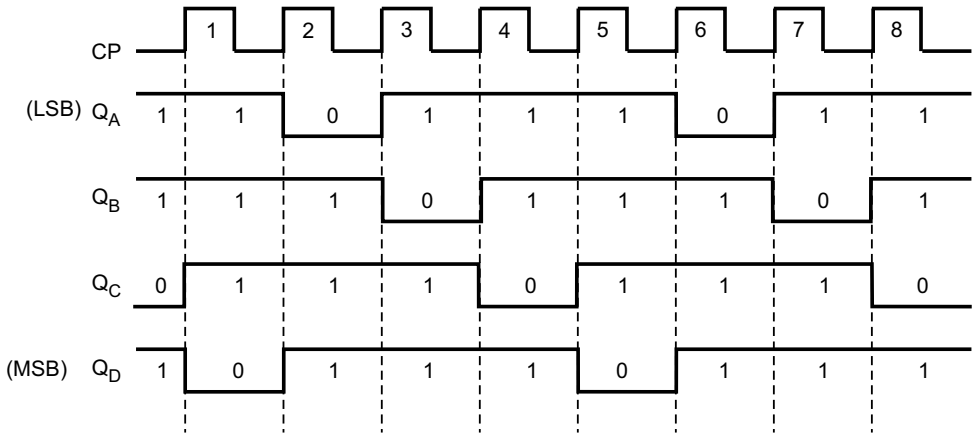


Fig. 7.6.5

Example 7.6.3 *Draw a six stage ring counter and explain its operation. Mention about the use of presetting the counter.* **Dec.-05, Marks 16**

Solution : The Fig. 7.6.6 shows the six stage ring counter. The counter is present to value $(000001)_2$ by setting bit 0 = 1 and remaining bits = 0.

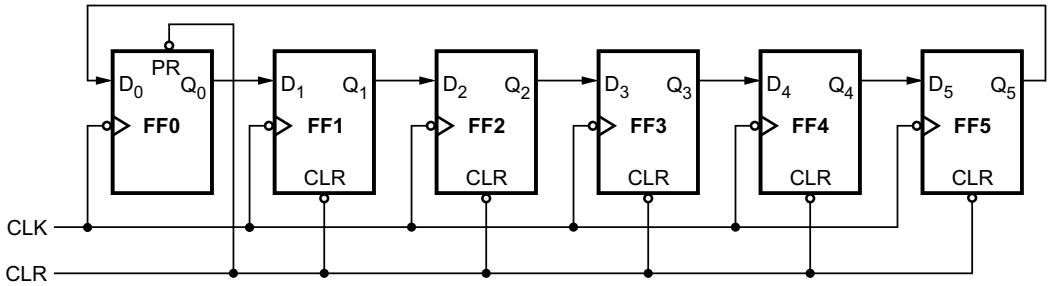


Fig. 7.6.6 Six stage ring counter

Operation : The Fig. 7.6.7 shows the operation of six-stage ring counter. On preset, FF0 (flip-flop 0) is set and FF1 to FF5 are reset. After each falling edge of the clock contents of ring counter are shifted 1 bit from LSB to MSB.

CLR	CLK	Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Operation
⊓	X	1	0	0	0	0	0	Preset
1	↓	0	1	0	0	0	0	**'1'** Bit follows a circular path to form ring counter
1	↓	0	0	1	0	0	0	
1	↓	0	0	0	1	0	0	
1	↓	0	0	0	0	1	0	
1	↓	0	0	0	0	0	1	
1	↓	1	0	0	0	0	0	

Fig. 7.6.7 Illustrating operation of six-stage ring counter

Waveform : (See Fig. 7.6.8 on next page.)

Review Questions

1. *Draw a six stage ring counter and explain its operation.* **Dec.-05, Marks 16**
2. *Draw the timing diagram of 4-bit ring counter.* **Dec.-04, Marks 2**
3. *Draw a 2-bit ripple counter and convert this into a 2-bit ring counter.* **Dec.-05, Marks 2**
4. *Explain the operation of shift and Ring counters.* **May-12, Marks 8**
5. *Design a 3 bit ring counter and find the mod of the designed counter.* **Dec.-12, Marks 2**

7.7 Johnson or Twisting Ringor Switch Tail Counter **May-07, Dec.-06,11,12**

In a Johnson counter, the Q output of each stage of flip-flop is connected to the D input of the next stage. The single exception is that the complement output of the last flip-flop is connected back to the D-input of the first flip-flop as shown in Fig. 7.7.1.

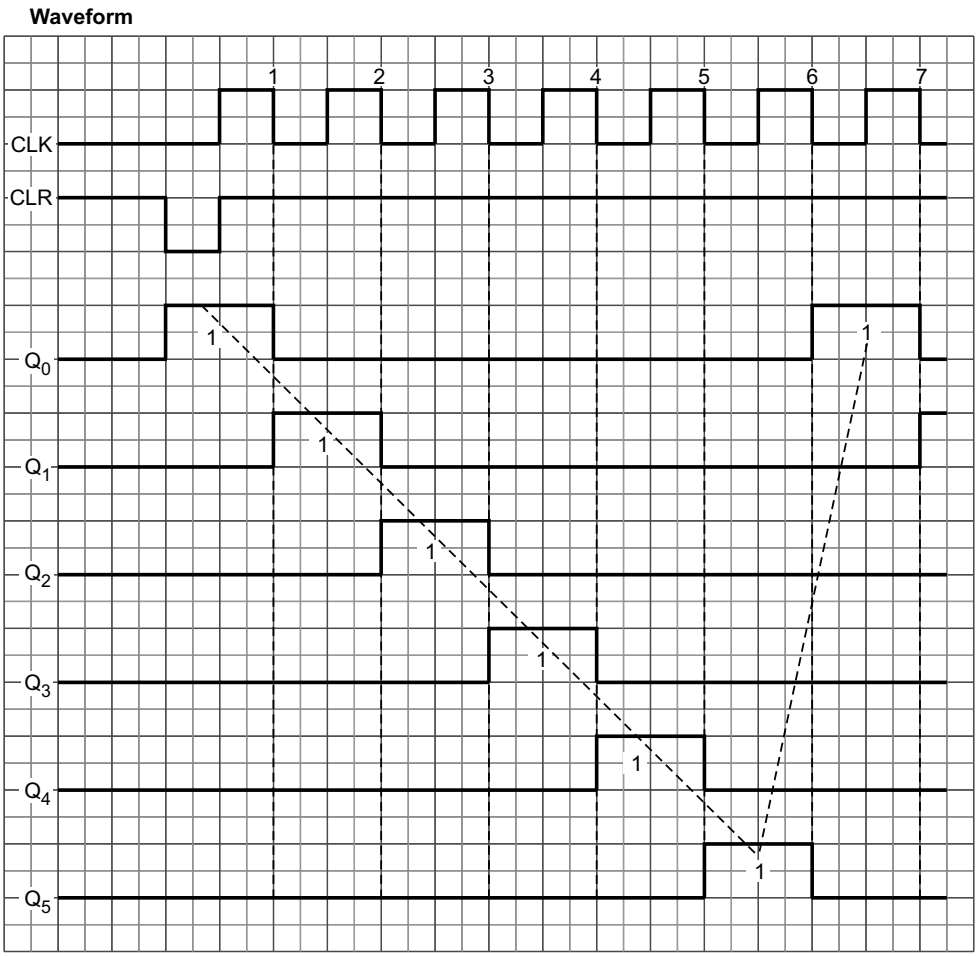


Fig. 7.6.8 Waveforms of a six-stage ring counter

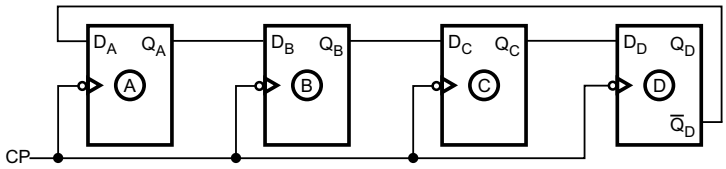


Fig. 7.7.1 Four-bit Johnson counter

Note : Johnson counter can be implemented with SR or JK flip-flops as well.

As shown in Fig. 7.7.1 there is a feedback from the rightmost flip-flop complement output to the leftmost flip-flop input. This arrangement produces a unique sequence of states.

Initially, the register (all flip-flops) is cleared. So all the outputs, Q_A, Q_B, Q_C, Q_D are zero. The output of last stage, Q_D is zero. Therefore complement output of last stage, $\overline{Q}_D$ is one. This is connected back to the D input of first stage. So D_A is one. The first falling clock edge produces Q_A= 1 and Q_B = 0,Q_C = 0, Q_D = 0 since D_B, D_C, D_D are zero. The next clock pulse produces Q_A= 1 ,Q_B = 1, Q_C = 0, Q_D = 0. The sequence of states is summarized in Table 7.7.1. After 8 states the same sequence is repeated.

Clock pulse	Q_A	Q_B	Q_C	Q_D
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1

Table 7.7.1 Four-bit Johnson sequence

In this case, four-bit register is used. So the four-bit sequence has a total of eight states. Fig. 7.7.2 gives the timing sequence for a four-bit Johnson counter.

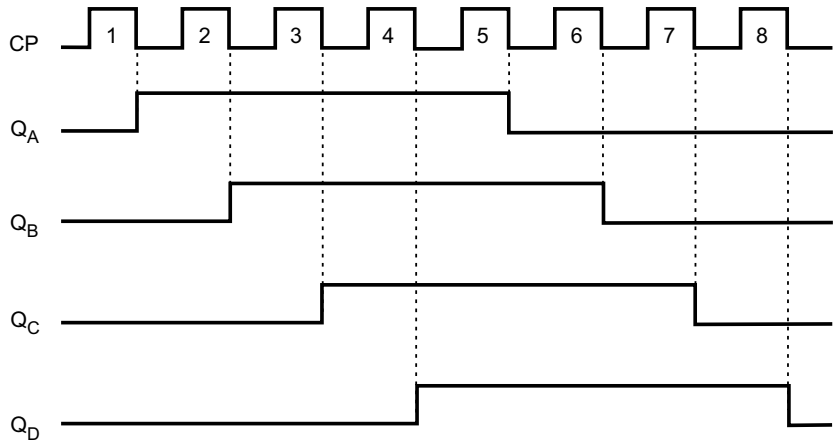


Fig. 7.7.2 Timing sequence for a four-bit Johnson counter

If we design a counter of five-bit sequence, it has a total of ten states, as shown in Table 7.7.2.

So in general we can say that, an n-stage Johnson counter will produce a modulus of $2 \times n$, where n is the number of stages (i.e. flip-flops) in the counter. Thus, Johnson counter requires only half the number of flip-flops compared to the standard ring counter. However, it requires more flip-flop than binary counter. As shown in tables, the counter will 'fill up'

Clock pulse	Q_A	Q_B	Q_C	Q_D	Q_E
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1

Table 7.7.2 Five-bit Johnson sequence

with 1s from left to right and then it will 'fill up' with 0s again. Another advantage of this type of sequence is that it is readily decoded with two input AND gates. Table 7.7.3 gives the count sequence and required decoding.

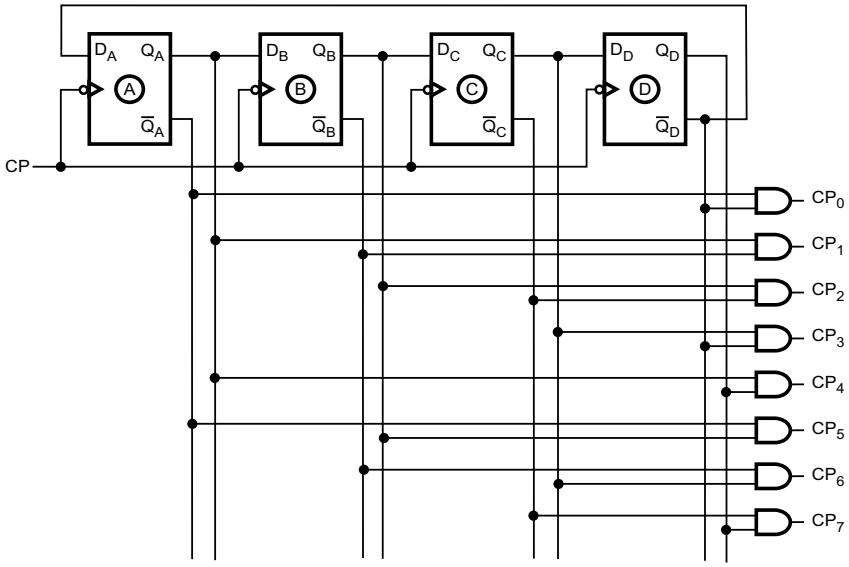


Fig. 7.7.3 Johnson counter with decoder

Clock pulse	Q_A	Q_B	Q_C	Q_D	AND Gate required for output
0	0	0	0	0	$\overline{Q}_A \overline{Q}_D$
1	1	0	0	0	$Q_A \overline{Q}_B$
2	1	1	0	0	$Q_B \overline{Q}_C$
3	1	1	1	0	$Q_C \overline{Q}_D$
4	1	1	1	1	$Q_A Q_D$
5	0	1	1	1	$\overline{Q}_A Q_B$
6	0	0	1	1	$\overline{Q}_B Q_C$
7	0	0	0	1	$\overline{Q}_C Q_D$

Table 7.7.3 Count sequence and required decoding

Example 7.7.1 *Design a 4-bit, 8-state Johnson counter using IC 74X194. Show how same counter can be modified as self correcting Johnson counter.*

Solution : Johnson counter is basically a twisted ring counter. The Fig. 7.7.4 (a) shows the basic circuit for a Johnson counter and Fig. 7.7.4 (b) shows its timing diagram. The Table 7.7.3 shows the states of a 4-bit Johnson counter.

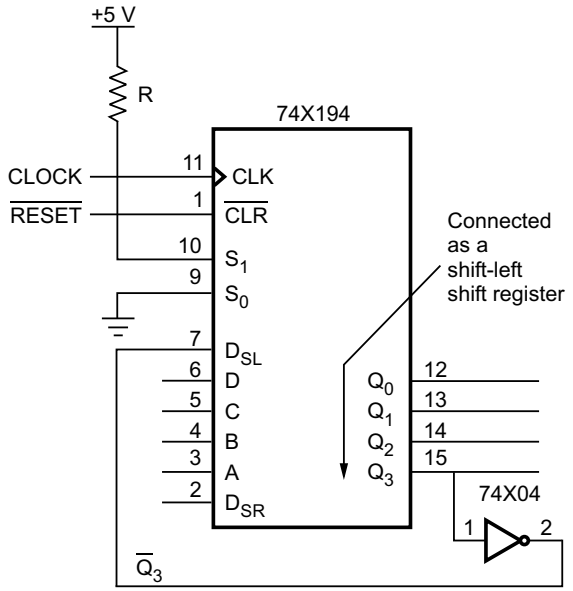


Fig. 7.7.4 (a) 4-bit Johnson counter

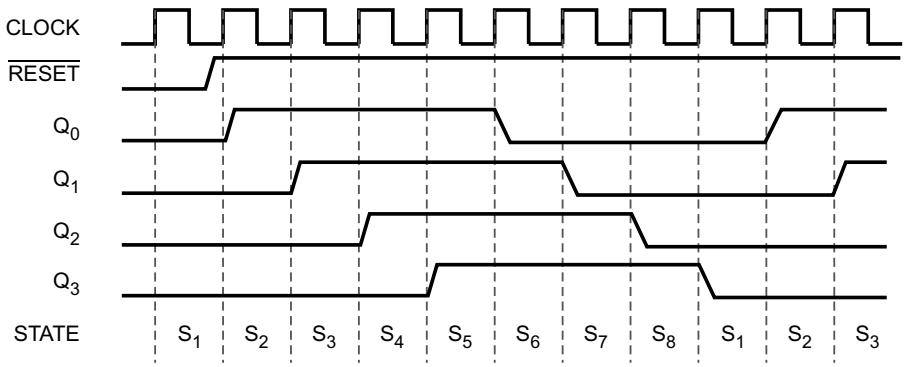


Fig. 7.7.4 (b) Timing diagram for a 4-bit Johnson counter

State Name	Q_3	Q_2	Q_1	Q_0
S_1	0	0	0	0
S_2	0	0	0	1
S_3	0	0	1	1
S_4	0	1	1	1
S_5	1	1	1	1
S_6	1	1	1	0
S_7	1	1	0	0
S_8	1	0	0	0

Table 7.7.4 States of a 4-bit Johnson counter

This counter can be modified to have self correcting Johnson counter as shown in the Fig. 7.7.5. Here, the connections are made such that circuit loads 0001 as the next state whenever the current state is 0XX0.

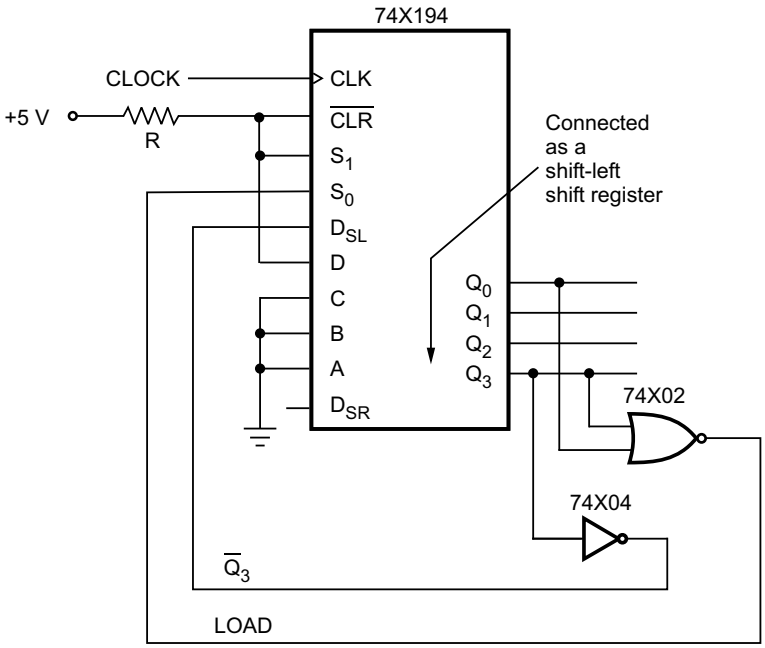


Fig. 7.7.5 Self-correcting 4-bit, 8-state Johnson counter

Example 7.7.2 *How many flip-flops are required to implement each of following in a Johnson counter configuration :*
i) Mod 10 ii) Mod 16

Solution : Johnson counter will produce a modulus of $2 \times n$ where n is the number of stages (i.e. flip-flops) in the counter. Therefore, Mod-10 requires 5 flip-flops and Mod-16 requires 8 flip-flops.

Example 7.7.3 *Draw a 5 flip-flop shift (Johnson) counter, its truth table and waveforms. Explain its operation as a decade counter.* **Dec.-05, Marks 16**

Solution : The Fig. 7.7.6 shows the 5-bit shift (Johnson) counter. Since this counter goes through 10 states, the frequency at the output of last flip-flop is $1/10^{th}$ of the clock frequency and hence it is a decade counter.

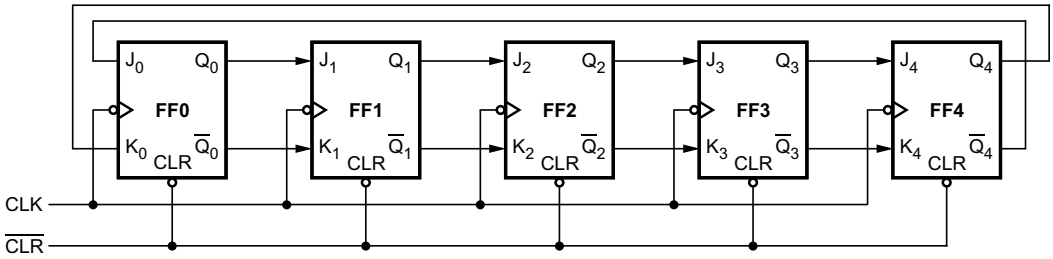


Fig. 7.7.6 5 Flip-flop shift (Johnson counter)

The Table 7.7.5 shows the truth table for the 5 flip-flop shift counter and illustrates its operation.

$\overline{CLR}$	CLK	Q_4	Q_3	Q_2	Q_1	Q_0	State	Decimal Equivalent
⊓	–	0	0	0	0	0	1	0
1	↓	0	0	0	0	1	2	1
1	↓	0	0	0	1	1	3	3
1	↓	0	0	1	1	1	4	7
1	↓	0	1	1	1	1	5	15
1	↓	1	1	1	1	1	6	31
1	↓	1	1	1	1	0	7	30
1	↓	1	1	1	0	0	8	28
1	↓	1	1	0	0	0	9	24
1	↓	1	0	0	0	0	10	16
1	↓	0	0	0	0	0	1	0

Table 7.7.5 Truth table

Waveform : (See Fig. 7.7.7 on next page.)

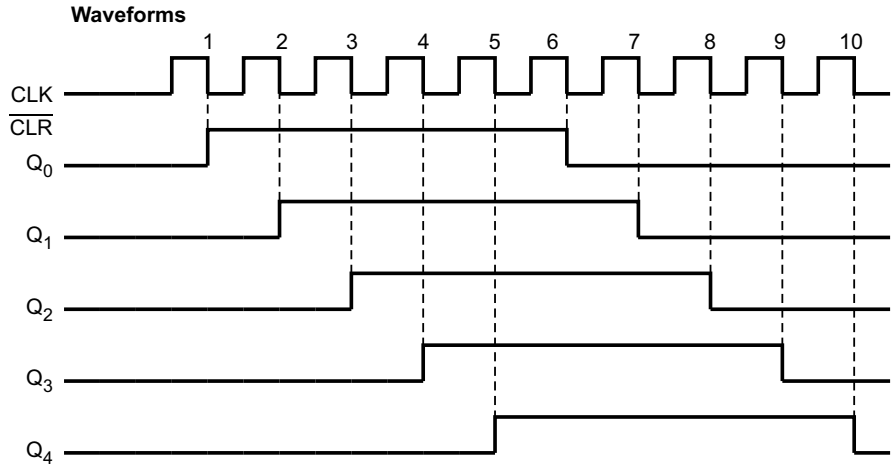


Fig. 7.7.7 Waveforms for 5 flip-flop shift counter

Review Questions

1. *Draw the 4-bit Johnson counter and explain the operation.* **May-07, Dec.-06, Marks 8**
2. *Design Johnson counter and state its advantages and disadvantages.* **Dec.-11, Marks 8**
3. *Design a 3 bit Johnson counter and explain its operation.* **Dec.-12, Marks 8**

7.8 Two Marks Questions with Answers

Q.1 ***Define registers.***

Ans. : A register is a group of flip-flops. So an n-bit register has a group of n flip-flops and is capable of storing any binary information/number containing n-bits.

Q.2 ***Define shift registers.*** **Dec.-10**

Ans. : The binary information in a register can be moved from stage to stage within the register or into or out of the register upon application of clock pulses. This type of bit movement or shifting is essential for certain arithmetic and logic operations used in microprocessors. This gives rise to group of registers called shift registers.

Q.3 ***What are the different types of shift type ? or***
Classify the registers with respect to serial and parallel input output.
Dec.-10, May-07

Ans. : There are five types. They are,

- Serial In Serial Out Shift Register
- Serial In Parallel Out Shift Register
- Parallel In Serial Out Shift Register
- Parallel In Parallel Out Shift Register
- Bidirectional Shift Register

Q.4 ***What is the difference between serial and parallel transfer ? What type of register is used in each case ?*** **Dec.-07**

Ans. : When data is transferred one bit at a time, the process of transfer is known as serial transfer. When multiple bits of data are transferred at a time, the process is known as parallel transfer. For parallel transfer, we can use parallel in and parallel out register. For serial transfer we can use left shift or right shift register.

Q.5 ***If a serial-in-serial-out shift register has N stages and if the clock frequency is f, what will be time delay between input and output ?*** **Dec.-03**

Ans. : The time delay between input and output is $T_D = \frac{N}{f}$.

Q.6 ***Draw the timing diagram of 4-bit ring counter.*** **Dec.-04**

Ans. : Refer section 7.6.

Q.7 ***Draw a 2-bit ripple counter and convert this into a 2-bit ring counter.*** **Dec.-05**

Ans. : Refer section 7.6.

Q.8 ***Define universal shift register.***

Ans. : The register which has both shifts (right-shift and left-shift) and parallel load capabilities is referred to as universal shift register.

Q.9 ***Mention applications of shift registers.***

Ans. :
1. Delay line
2. Serial to parallel converter
3. Parallel to serial converter.
4. Pseudo-random binary sequence generator.
5. Sequence detector.

Q.10 ***What is shift register counter ?***

Ans. : A shift register can also be used as a counter. A shift register with the serial output connected back to the serial input is called **shift register counter**.

❑❑❑

8 Analysis and Design of Synchronous Sequential Circuits

Syllabus

State diagram - State table - State minimization - State assignment - Excitation table and maps - Circuit implementation. Synchronous Sequential Circuits : General model - Classification - Design - Use of algorithmic state machine - Analysis of synchronous sequential circuits.

Contents

8.1 *Introduction* . . . . . **May-06, Dec.-09,** · · · · · · · · · · · Marks 4

8.2 *Clocked Sequential Circuits* **May-05, 09,**

. . . . . **Dec.-07, 08, 10,** · · · · · · · · · · · · Marks 4

8.3 *Analysis of Clocked Sequential Circuits* . . . . . **Dec.-03, 06, 07, May-12,** · · · · · Marks 8

8.4 *Design of Clocked Sequential Circuits* **Dec.03, 05, 06, 10,**

. . . . . **May-09, 10,** · · · · · · · · · · · · · · Marks 10

8.5 *Sequence Generator*

8.6 *Sequence Detector* . . . . . **Dec.-12,** · · · · · · · · · · · · · · · · · · Marks 8

8.7 *Simplification of Incompletely* **May-12**

Specified Machines . . . . . .

8.8 *Algorithmic State Machine* . . . . . **May-11, 12,** · · · · · · · · · · · · · · · · Marks 2

8.9 *Two Marks Questions with Answers*

8.1 Introduction

May-06, Dec.-09

We have already introduced to synchronous sequential circuits. These circuits are further classified depending on the timing of their signals : **Synchronous sequential circuits** and **Asynchronous sequential circuits**. In synchronous sequential circuits, signals can affect the memory elements only at discrete instants of time. In asynchronous sequential circuits change in input signals can affect memory element at any instant of time.

Sr. No.	Synchronous sequential circuits	Asynchronous sequential circuits
1.	In synchronous circuits, memory elements are clocked flip-flops.	In asynchronous circuits, memory elements are either unclocked flip-flops or time delay elements.
2.	In synchronous circuits, the change in input signals can affect memory element upon activation of clock signal.	In asynchronous circuits change in input signals can affect memory element at any instant of time.
3.	The maximum operating speed of clock depends on time delays involved.	Because of absence of clock, asynchronous circuits can operate faster than synchronous circuits.
4.	Easier to design.	More difficult to design.

Table 8.1.1 Comparison between synchronous and asynchronous sequential circuits

In this chapter, we are going to study the analysis and design of synchronous sequential circuits and design of synchronous counters.

Review Questions

1. *Differentiate between synchronous and asynchronous sequential circuits.* May-06, Marks 4
2. *Distinguish between synchronous and asynchronous sequential circuits.* Dec.-09, Marks 2

8.2 Clocked Sequential Circuits

May-05, 09; Dec.-07, 08, 10

In synchronous or clocked sequential circuits, clocked flip-flops are used as memory elements, which change their individual states in synchronism with the periodic clock signal. Therefore, the change in states of flip-flops and change in state of the entire circuit occurs at the transition of the clock signal. The states of the output of the flip-flop in the sequential circuit gives the state of the sequential circuit.

Present state : The status of all state variables, at some time t, before the next clock edge, represents condition called **present state**.

Next state : The status of all state variables, at some time, t + 1, represents a condition called **next state**.

The synchronous or clocked sequential circuits are represented by two models.

- **Moore model :** The output depends only on the present state of the flip-flops.
- **Mealy model :** The output depends on both the present state of the flip-flop(s) and on the input (s).

8.2.1 Moore Model

As mentioned earlier, when the output of the sequential circuit depends only on the present state of the flip-flop, the sequential circuit is referred to as **Moore model.** Let us see one example of Moore model. Fig. 8.2.1 shows a sequential circuit which consists of two JK flip-flops and AND gate. The circuit has one input X and one output Y.

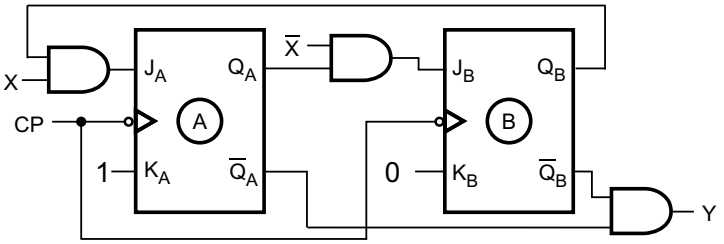


Fig. 8.2.1 Example of Moore model

As shown in the Fig. 8.2.1, input is used to determine the inputs of the flip-flops. It is not used to determine the output. The output is derived using only present states of the flip-flops or combination of it (in this case $Y = Q_A Q_B$).

In general form the Moore model can be represented with its block schematic as shown in Fig. 8.2.2 (a) and (b).

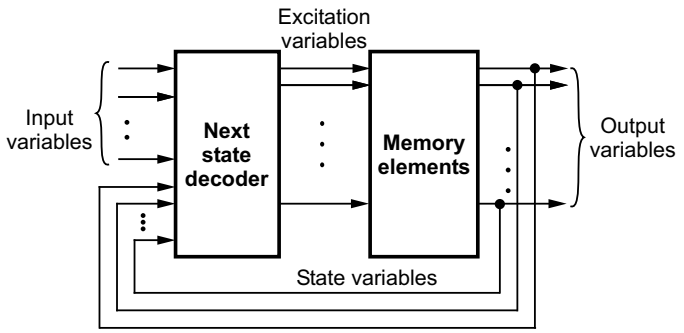


Fig. 8.2.2 (a) Moore model

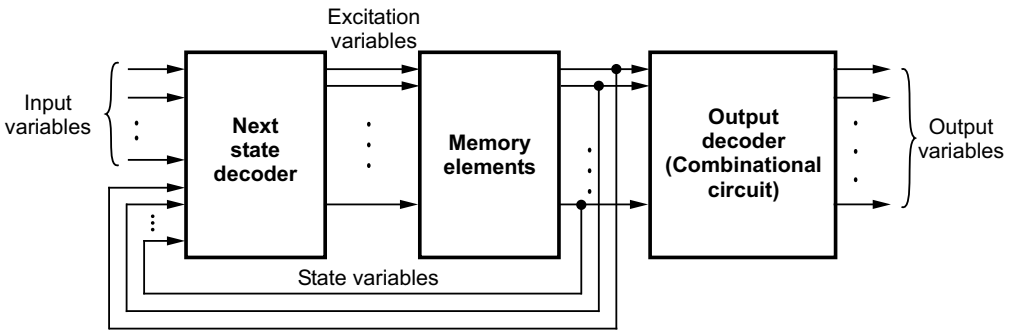


Fig. 8.2.2 (b) Moore circuit model with an output decoder

In the Moore model, as output depends only on present state of flip-flops, it appears only after the clock pulse is applied, i.e. it varies in synchronism with the clock input.

8.2.2 Mealy Model

When the output of the sequential circuit depends on both the present state of flip-flop(s) and on the input(s), the sequential circuit is referred to as **Mealy model**. Fig. 8.2.3 shows the sample Mealy model. As shown in the Fig. 8.2.3, the output of the circuit is derived from the combination of present state of flip-flops and input (s) of the circuit.

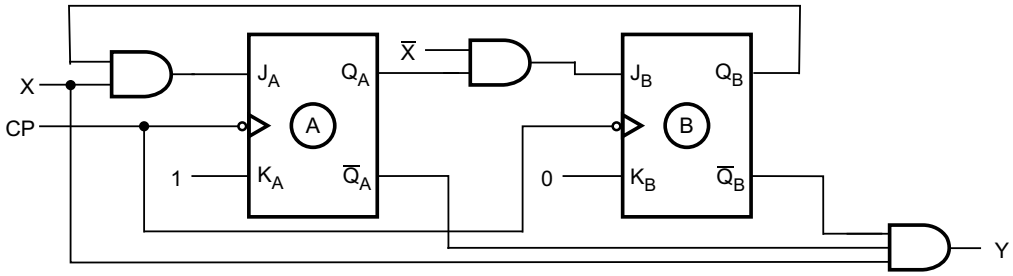


Fig. 8.2.3 Example of Mealy model

Looking at Fig. 8.2.3, we can easily realize that, changes in the input within the clock pulses can not affect the state of the flip-flop. However, they can affect the output of the circuit. Due to this, if the input variations are not synchronized with the clock, the derived output will also not be synchronized with the clock and we get false output (as it is a synchronous sequential circuit). The false outputs can be eliminated by allowing input to change only at the active transition of the clock (in our example HIGH-to-LOW). In general form the Mealy model can be represented with its block schematic as shown in Fig. 8.2.4.

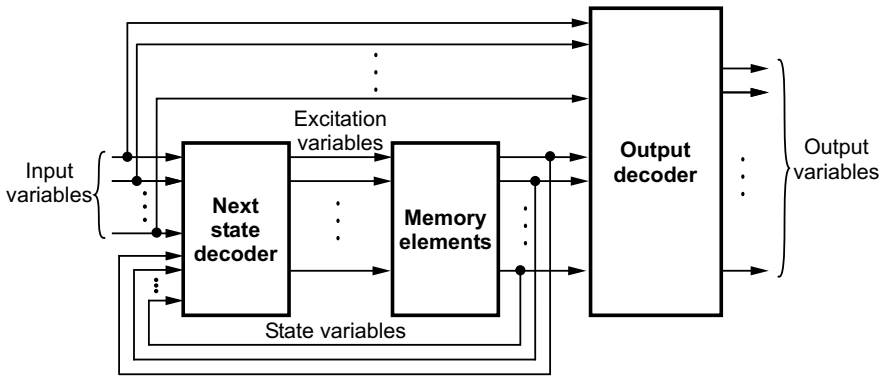


Fig. 8.2.4 Mealy circuit model

8.2.3 Moore vs Mealy Circuit Models

Sr. No.	Moore Model	Mealy Model
1.	Its output is a function of present state only.	Its output is a function of present state as well as present input.
2.	Input changes does not affect the output.	Input changes may affect the output of the circuit.
3.	Moore model requires more number of states for implementing same function.	It requires less number of states for implementing same function.

Table 8.2.1

8.2.4 Representations of Sequential Circuits

8.2.4.1 State Diagram

State diagram is a pictorial representation of a behaviour of a sequential circuit. Fig. 8.2.5 shows a state diagram. The state is represented by the circle, and the transition between states is indicated by directed lines connecting the circles. A directed line connecting a circle with itself indicates that next state is same as present state. The binary number inside each circle identifies the state represented by the circle. The directed lines are labelled with two binary numbers separated by a symbol '/'. The input value that causes the state transition is labelled first and the output value during the present state is labelled after the symbol '/'.

In case of Moore circuit, the directed lines are labelled with only one binary number representing the state of the input that causes the state transition. The output state is indicated within the circle, below the present state because output state depends only on present state and not on the input. Fig. 8.2.6 shows the state diagram for Moore circuit.

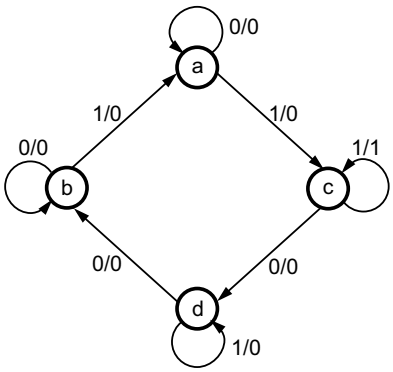


Fig. 8.2.5 State diagram for Mealy circuit

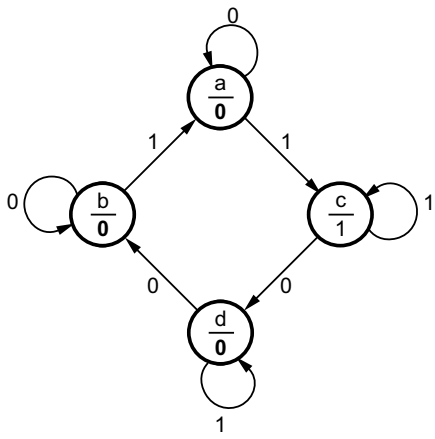


Fig. 8.2.6 State diagram for Moore circuit

8.2.4.2 State Table

Although the state diagram provides a description of the behaviour of a sequential circuit that is easy to understand, to proceed with the implementation of the circuit, it is convenient to translate the information contained in the state diagram into a tabular form called **state synthesis table** or simply **state table**. Table 8.2.2 (a) shows the state table for the state diagram shown in Fig. 8.2.5. It represents relationship between input, output and flip-flop states. It consists of three sections labeled present state, next state, and output. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state is state of the flip-flop after the application of a clock pulse, and the output section gives the values of the output variables during the present state. Both the next state and output sections have two columns representing two possible input conditions : X = 0 and X = 1.

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
AB	AB	AB	Y	Y
a	a	c	0	0
b	b	a	0	0
c	d	c	0	1
d	b	d	0	0

Table 8.2.2 (a)

Present state	Next state		Output
	X = 0	X = 1	Y
AB	AB	AB	
a	a	c	0
b	b	a	0
c	d	c	1
d	b	d	0

Table 8.2.2 (b)

In case of Moore circuit the output section has only one column since output does not depend on input. The Table 8.2.2 (b) shows the state table for Moore circuit whose state diagram is shown in Fig. 8.2.6.

8.2.4.3 Transition Table

A transition table takes the state table one step further. The state diagram and state table represent state using symbols or names. In the transition table specific state variable values are assigned to each state. Assignment of values to state variables is called **State assignment**. Like state table transition table also represents relationship between input, output and flip-flop states. The Fig. 8.2.7 shows the transition table.

Present state		Next state		Output	
		X = 0	X = 1	X = 0	X = 1
A	B	AB	AB	Y	Y
0	0	0 0	1 0	0	1

0	1	1 1	0 0	0	0
1	0	1 0	0 1	1	0
1	1	0 0	1 0	1	0

Fig. 8.2.7

Here, AB are the state variables. The AB = 00 represents one state, AB = 01 represents second state and so on.

8.2.5 Conversion of Models

Rules to convert Mealy to Moore Model

1. If all the transitions in a Mealy model to a particular state are associated with only one type of output (either 0 or 1) then in corresponding Moore model that output becomes state output. This is illustrated in Fig. 8.2.8. Here, T_1, T_2 and T_3 are the paths leading to state a and corresponding outputs are all zeroes. Thus we can say state a output is 0.

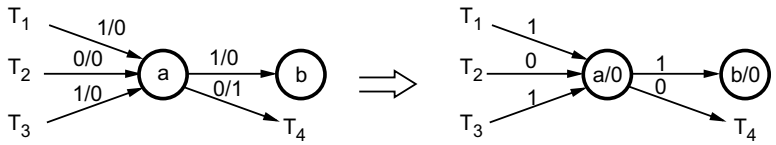


Fig. 8.2.8 Rule 1: Mealy to Moore conversion

When input is 1, state a leads to state b and there is no other input path. Thus we can say state b output is 0 (the corresponding output of input path).

2. If the outputs of all transitions in a Mealy model to a particular state are not same we need to insert intermediate state variables. This is illustrated in Fig. 8.2.9.

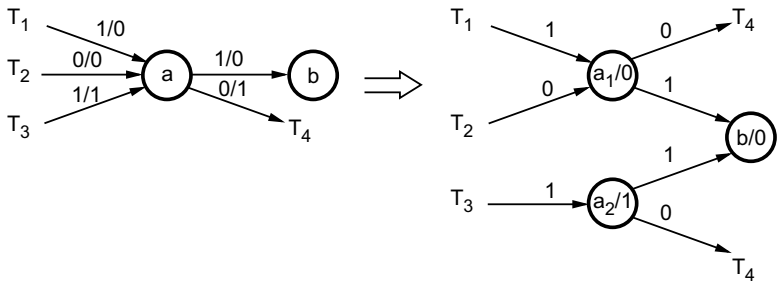
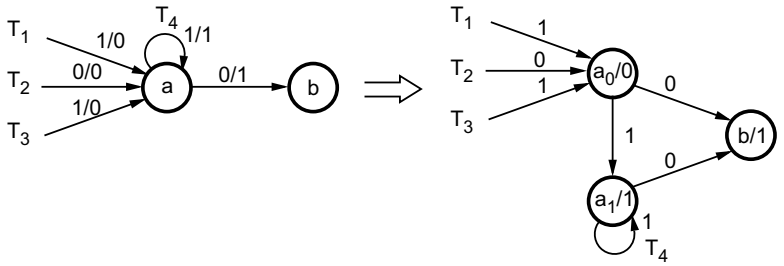


Fig. 8.2.9 Rule 2 : Mealy to Moore conversion

Let us see some examples of converting Mealy circuit to Moore circuit.

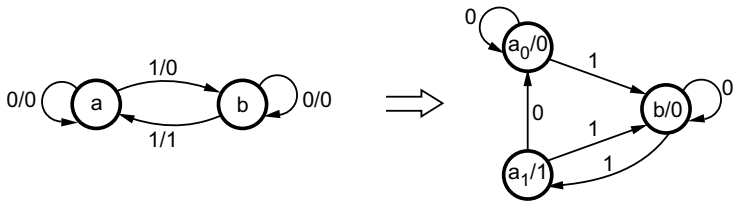
Example 1



(a) Mealy state diagram **(b) Moore state diagram**

Fig. 8.2.10

Example 2

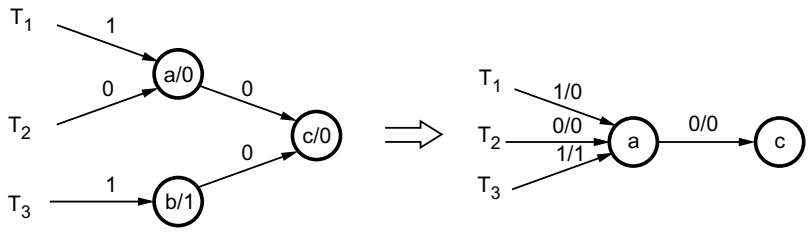


(a) Mealy state diagram **(b) Moore state diagram**

Fig. 8.2.11

Rules to convert Moore to Mealy model

1. If the state transition from two different states of the same input leads to common state then one state can be eliminated. This is illustrated in Fig. 8.2.12.



(a) Moore circuit **(b) Mealy circuit**

Fig. 8.2.12

Here, states a and b leads to common state c when input is 0. Therefore, we can eliminate state a or b.

2. If the state transition from two different states of the same input does not lead to same state then state output becomes the output corresponds to each input transition of that state.

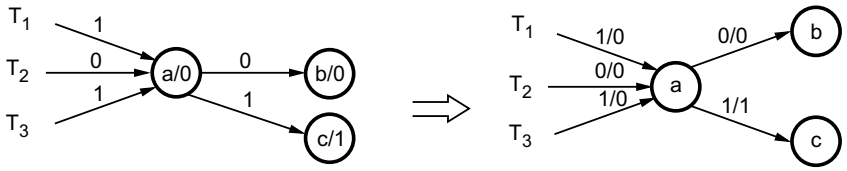


Fig. 8.2.13

Review Questions

1. *What is a state ?* **May-09, Marks 2**
2. *Define present state and next state.*
3. *Draw and explain the block diagram of Moore model.*
4. *What is Moore machine ?* **Dec.-07, Marks 2**
5. *Draw and explain the block diagram of Mealy model.*
6. *What is a Mealy machine ? Give an example.* **May-05, Dec.-08, Marks 2**
7. *Compare Moore and Mealy circuits.* **May-05, Dec.-08, 10, May-09, Marks 4**
8. *What is state diagram ? Explain with the help of example.*
9. *What is state table ? Explain with the help of example.*
10. *What is transition table ?*
11. *Explain the rules to convert Mealy to Moore model.*
12. *Explain the rules to convert Moore to Mealy model.*

8.3 Analysis of Clocked Sequential Circuits

Dec.-03, May-12

The behaviour of a sequential network is determined from the inputs, the outputs, and the states of its flip-flops. Both the outputs and the next state are function of the inputs and the present state (in case of Moore circuits the outputs are function of only present state). The analysis of sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states. The success of analysis or design of sequential network depends largely on the aids and systematic techniques such as transition tables, state tables, state diagrams and state equations used in these processes.

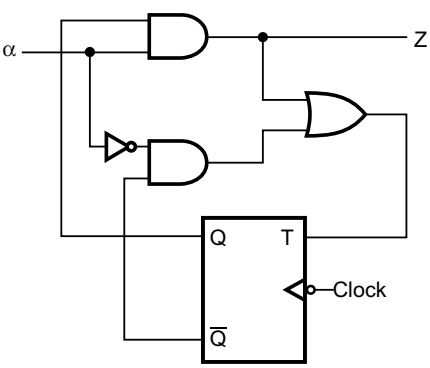


Fig. 8.3.1

Consider the sequential circuit to be analysed as shown in Fig. 8.3.1.

Let us see the steps to analyze the given synchronous sequential circuit.

1. Determine the flip-flop input equations and the output equations from the sequential circuit.

$$Z = \alpha Q$$

$$T = \alpha Q + \bar{\alpha}\,\bar{Q}$$

2. Derive the transition equation.

The transition equation for T flip-flop is

$$Q^+ = T \oplus Q$$

$$Q^+ = \left(\alpha Q + \bar{\alpha}\,\bar{Q}\right) \oplus Q$$

3. Plot the next step map for each flip-flop

$$Q^+ = \left(\alpha Q + \bar{\alpha}\,\bar{Q}\right) \oplus Q$$

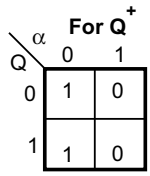


Fig. 8.3.2

4. Plot the transition table

Present state	Next state		Output	
	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$
Q	Q^+	Q^+	Z	Z
0	1	0	0	0
1	1	0	0	1

Table 8.3.1 Transition table

Note : If circuit consists of more than one flip-flop we have to combine the map of flip-flop to derive the transition table. (Refer next example)

5. Draw the state table

The transition table shown in Table 8.3.1 can be converted into state table as shown in Table 8.3.2. Here, new symbols to binary codes are assigned. They are a = 0, b = 1.

Present state	Next state		Output	
	$\alpha = 0$	$\alpha = 1$	$\alpha = 0$	$\alpha = 1$
a(0)	b	a	0	0
b(1)	b	a	0	1

Table 8.3.2 State table

6. Draw state diagram

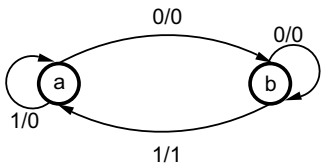


Fig. 8.3.3 State diagram

Examples for Understanding

Example 8.3.1 *Construct the transition table, state table and state diagram for the Moore sequential circuit given below :*

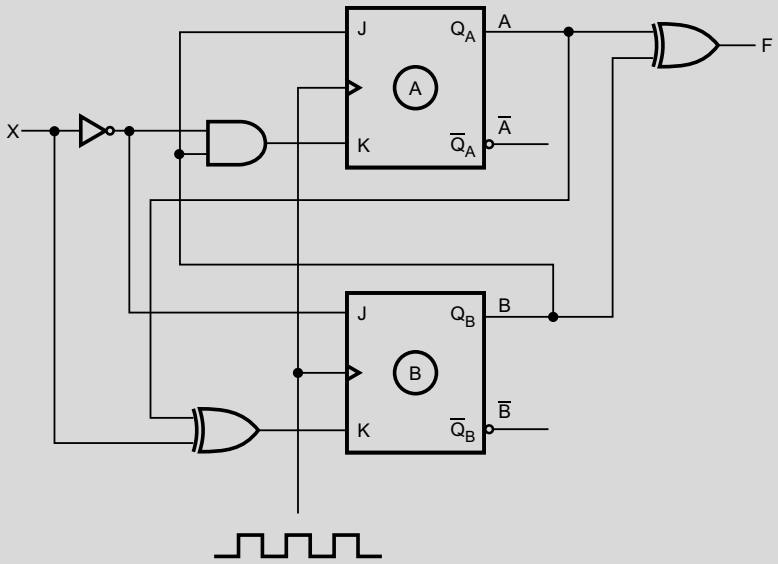


Fig. 8.3.4 (a)

Solution :

1. Determine the flip-flop input equations and the output equations from the sequential circuit.

$$F = A \oplus B$$

$$J_A = B \qquad K_A = \overline{X}B$$

$$J_B = \overline{X} \qquad K_B = X \oplus A$$

2. Derive the transition equations.

The transition equations for JK flip-flops can be derived from the characteristic equation of JK flip-flop as follows :

We know that for JK flop-flop

$$Q^+ = J\overline{Q} + \overline{K}Q$$

$$\therefore \; A^+ = Q_A^+ = J_A \overline{Q}_A + \overline{K}_A Q_A = B\overline{Q}_A + \overline{\overline{X}B}\, Q_A = B\overline{Q}_A + \left(X + \overline{B}\right) Q_A$$

$$= B\overline{A} + \left(X + \overline{B}\right) A$$

and $$B^+ = Q_B^+ = J_B \overline{Q}_B + \overline{K}_B Q_B = \overline{X}\,\overline{Q}_B + \overline{X \oplus A}\; Q_B$$

$$= \overline{X}\,\overline{B} + \overline{X \oplus A} \cdot B$$

3. Plot a next-state maps for each flip-flop.

The next-state maps are :

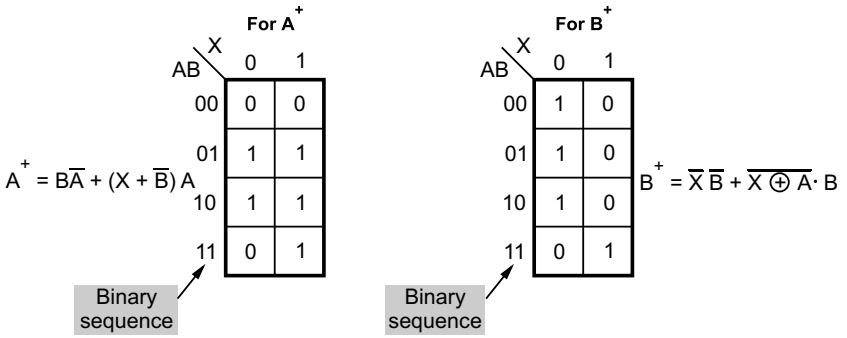


Fig. 8.3.4 (b)

4. Plot the transition table.

The transition table can be formed by combining the above two maps. The Table 8.3.3 shows the transition table.

Present state		Next state				Output
		X = 0		X = 1		
A	B	A^+	B^+	A^+	B^+	F = (A ⊕ B)
0	0	0	1	0	0	0
0	1	1	1	1	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0

Table 8.3.3 Transition table

Note : For Moore sequential circuit output only depends on present state and not on the input.

5. Draw the state table

By assigning a = 0 0, b = 01, c = 10 and d = 11 we can write state table from the transition table as shown.

Present state	Next state		Output
A B	X = 0	X = 1	F
	A^+B^+	A^+B^+	
a	b	a	0
b	d	c	1
c	d	c	1
d	a	d	0

Table 8.3.4

6. Draw state diagram

From the state table we can draw state diagram as shown in Fig. 8.3.5.

Note : In case of Moore model, the directed lines are labelled with only one binary number representing the state of the input that causes the state transition. The output state is indicated within the circle, below the present state because output state depends only on present state and not on the input.

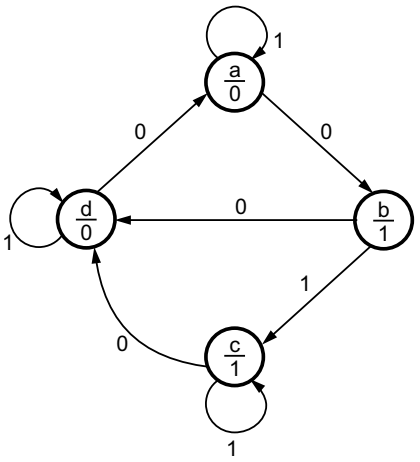


Fig. 8.3.5

Example 8.3.2 *A sequential circuit with 2D FFs A and B and input X and output Y is specified by the following next state and output equations.*

$A(t+1) = AX + BX$

$B(t+1) = A'X$

$Y = (A + B)X'$

i) Draw the logic diagram of the circuit.

ii) Derive the state table.

iii) Derive the state diagram.

Dec.-06, May-07, Marks 16

Solution :

i) Logic diagram

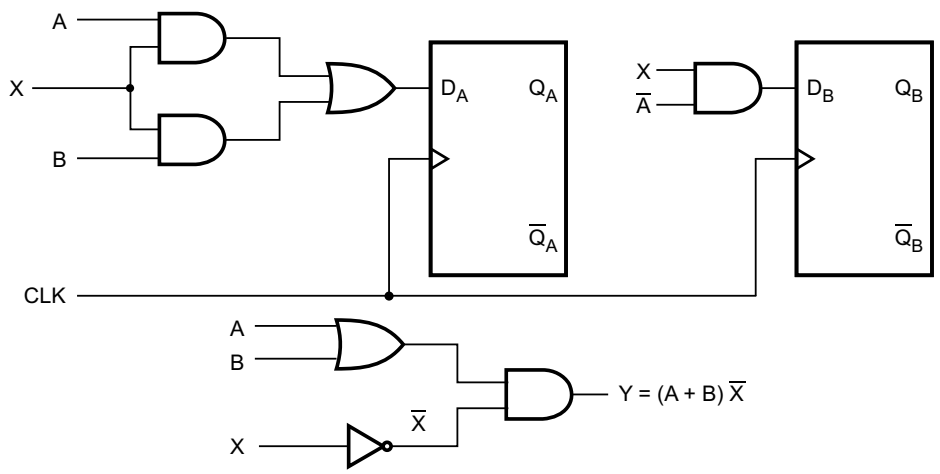


Fig. 8.3.6 (a)

ii) State table :

Step 1 : Plot the next-state map for each flip-flop.

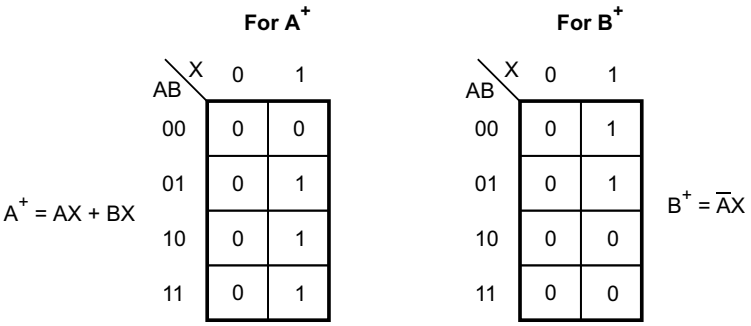


Fig. 8.3.6 (b)

Step 2 : Plot the transition table.

Present state		Next state				Output	
		X = 0		X = 1		X = 0	X = 1
A	B	A^+	B^+	A^+	B^+	$Y = (A + B)\,\overline{X}$	
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

Step 3 : Draw the state table.

By assigning a = 00, b = 01, c = 10 and d = 11 we can write state table from the transition table.

Present state	Next state		Output Y	
	X = 0	X = 1	X = 0	X = 1
	$A^+ B^+$	$A^+ B^+$		
a	a	b	0	0
b	a	d	1	0
c	a	c	1	0
d	a	c	1	0

iii) State diagram

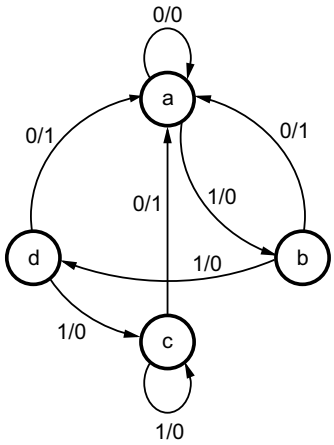


Fig. 8.3.7

Example 8.3.3 *A sequential circuit has four FFs ABCD and an input X is described by the following state equations.*

$A\ (t + 1) = (C\overline{D} + \overline{C}D)X + (CD + \overline{C}\overline{D})\overline{X}$

$B\ (t + 1) = A$

$C\ (t + 1) = B$

$D\ (t + 1) = C$

i) Obtain the sequence of states when X = 1, starting from state ABCD = 0001.

ii) Obtain the sequence of states when X = 0 starting from state ABCD = 0000.

Dec.-06, May-07, Marks 16

Solution :

i) X = 1 and Initially ABCD = 0001.

A B C D ⇐ 1000
0100
0010
1001

A B C D ⇐ 1000, 0100, 0010, 1001, 1100, 0110,
1011, 0101, 1010, 1101, 1110, 1111,
0111, 0011, 0001

ii) X = 0 and Initially ABCD = 0000

A B C D ⇐ 1000, 1100, 1110, 0111, 1011, 1101,
0110, 0011, 1001, 0100, 1010, 0101,
0010, 0001, 0000

Example 8.3.4 *A sequential circuit has one flip-flop Q, two inputs x and y, and one output S. It consists of a full adder circuit connected to a D flip-flop, as shown below. Derive the state table and state diagram of the sequential circuit.* **Dec.-07, Marks 12**

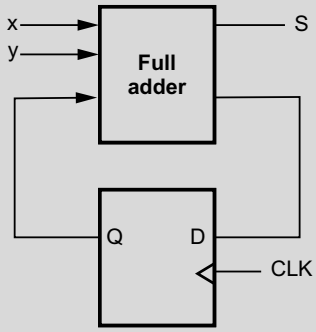


Fig. 8.3.8

Solution :

Step 1 : Derive the state table

Present state	Next state	Output
x y Q	Q+	S
0 0 0	0	0
0 0 1	0	1
0 1 0	0	1
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	1	0
1 1 1	1	1

Table 8.3.5 State table

Step 2 : Draw state diagram

State : a = 0, b = 1

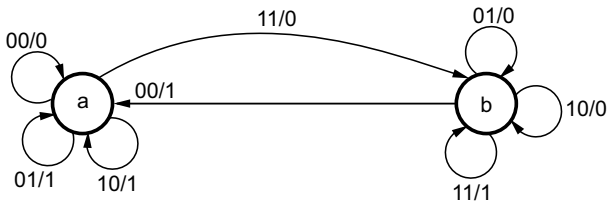


Fig. 8.3.9 State diagram

Example 8.3.5 *For the circuit shown in Fig. 8.3.10, write down the state table and draw the state diagram and analyze the operation.* **May-10, Marks 16**

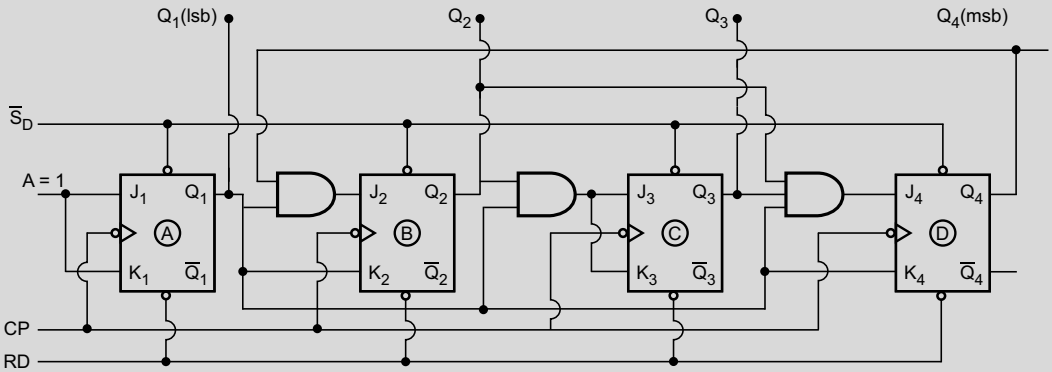


Fig. 8.3.10

Solution :

Step 1 : Inputs for each flip-flop

$J_1 = 1$ $K_1 = 1$

$J_2 = Q_1 Q_4$ $K_2 = Q_1$

$J_3 = Q_2 Q_1$ $K_3 = Q_2 Q_1$

$J_4 = Q_1 Q_2 Q_3$ $K_3 = Q_1$

Step 2 : Transition equations for each flip-flop

For JK flip-flop $Q^+ = J\overline{Q} + \overline{K}Q$

$\therefore \quad Q_1^+ = J_1 \overline{Q}_1 + \overline{K}_1 Q_1 = \overline{Q}_1 + 0 = \overline{Q}_1$

$\therefore \quad Q_2^+ = Q_1 Q_4 \overline{Q}_2 + \overline{Q}_1 Q_2$

$\therefore \quad Q_3^+ = Q_1 Q_2 \overline{Q}_3 + \overline{Q_1 Q_2} Q_3 = Q_1 Q_2 \overline{Q}_3 + (\overline{Q}_1 + \overline{Q}_2) Q_3$

$\therefore \quad Q_4^+ = Q_1 Q_2 Q_3 \overline{Q}_4 + \overline{Q}_1 Q_4$

Step 3 : Next state maps for each flip-flop

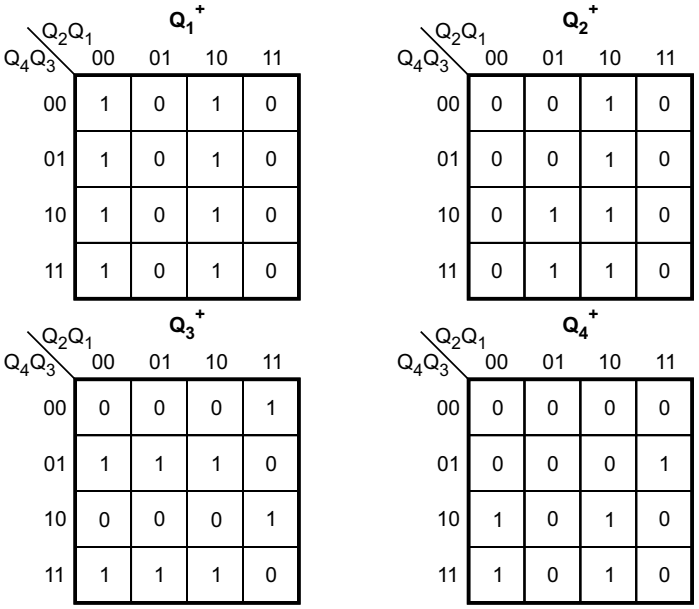


Fig. 8.3.11

Step 4 : Transition table

Present state				Next state			
Q_4	Q_3	Q_2	Q_1	Q_4^+	Q_3^+	Q_2^+	Q_1^+
0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	1	0
1	0	1	0	1	0	1	1
1	0	1	1	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	0	0	0

Step 5 : State assignment

a = 0000, b = 0001, c = 0010, d = 0011, e = 0100, f = 0101, g = 0110, h = 0111, i = 1000, j = 1001, k = 1010, l = 1011, m = 1100, n = 1101, o = 1110, p = 1111

Step 6 : State table

Present state	Next state
a	b
b	a
c	d
d	e
e	f
f	e
g	h
h	i
i	j
j	c
k	l
l	e
m	n
n	g
o	p
p	a

Step 7 : State diagram

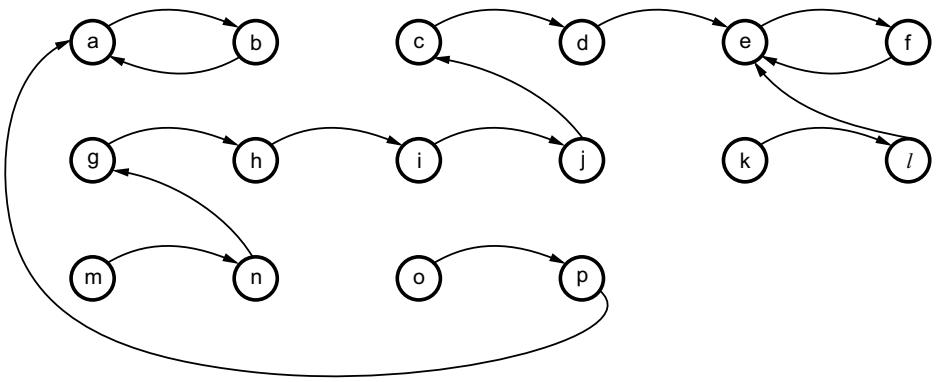


Fig. 8.3.12

Examples for Practice

Example 8.3.6 *Derive the state diagram for the given circuit.*

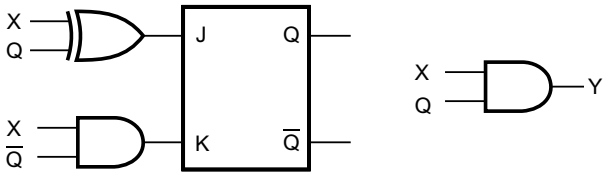


Fig. 8.3.13

Example 8.3.7 *Give output function, transition table and state diagram by analyzing the sequential circuit shown in Fig. 8.3.14.*

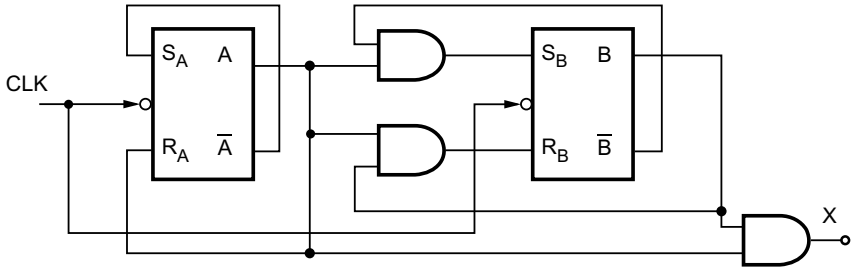


Fig. 8.3.14

Review Questions

1. *Explain the various steps in the analysis of synchronous sequential circuits with suitable example.* **Dec.-03, Marks 8**
2. *Describe the method to analyze the behavior of a synchronous sequential circuit.* **May-12, Marks 8**

8.4 Design of Clocked Sequential Circuits

Dec.03, 05, 06, 10; May-09, 10

The recommended steps for the design of a clocked synchronous sequential circuit are as follows :

1. It is necessary to first obtain the state table from the given circuit information such as a state diagram, a timing-diagram, or other pertinent information.
2. The number of states may be reduced by state reduction technique if the sequential circuit can be categorized by input-output relationships independent of the number of states.
3. Assign binary values to each state in the state table.
4. Determine the number of flip-flops needed and assign a letter symbol to each.
5. Choose the type of flip-flop to be used.
6. From the state table, derive the circuit excitation and output tables.
7. Using the K-map or any other simplification method, derive the circuit output functions and the flip-flop input functions.
8. Draw the logic diagram.

Let us see the details of various steps involved in the design of clocked sequential circuits.

8.4.1 State Reduction

The state reduction technique basically avoids the introduction of redundant states. The reduction in redundant states reduce the number of required flip-flops and logic gates, reducing the cost of the final circuit. The two states are said to be redundant or equivalent, if every possible set of inputs generate exactly same output and same next state. When two states are equivalent, one of them can be removed without altering the input-output relationship.

Let us illustrate the state reduction technique with an example. We start with a sequential circuit whose specification is given in the state diagram of Fig. 8.4.1. As shown in the diagram, the states are denoted by letter symbols instead of their binary values, because in state reduction technique internal states are not important; but only input-output sequences are important.

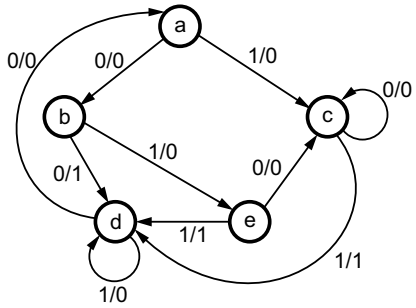


Fig. 8.4.1 State diagram

Step 1 : Determine the state table for given state diagram

Table 8.4.1 shows the state table for given state diagram

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	c	0	0
b	d	e	1	0
c	c	d	0	1
d	a	d	0	0
e	c	d	0	1

Table 8.4.1 State table

Step 2 : Find equivalent states

As mentioned earlier, in equivalent states every possible set of inputs generate exactly same output and same next state. In the given circuit there are two input combinations : X = 0 and X = 1. Looking at the state table for two present states that go to the same next state and have the same output for both input combinations, we can easily find that states c and e are equivalent. This is because, c and e both states go to states c and d and have outputs of 0 and 1 for X = 0 and X = 1, respectively. Therefore, state e can be

removed and replaced by c. The final reduced table is shown in Table 8.4.2. The state diagram for the reduced table consists of only four states and is shown in Fig. 8.4.2.

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	b	c	0	0
b	d	c	1	0
c	c	d	0	1
d	a	d	0	0

Table 8.4.2 Reduced state table

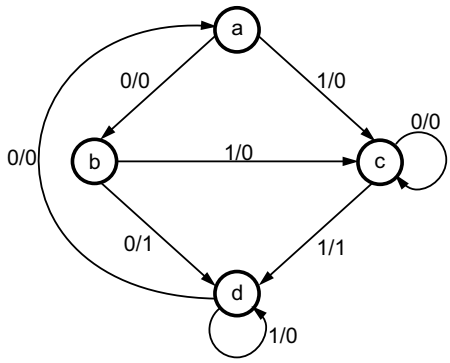


Fig. 8.4.2 Reduced state diagram

Examples for Understanding

Example 8.4.1 *Minimize the state table shown given below.* **Dec.-03, Marks 4**

Present state	Next state, Z (Output)	
	X (Input)	
	0	1
A	B,0	C,0
B	B,0	D,0
C	B,0	C,0
D	E,1	C,0
E	B,0	D,0

Solution :

Step 1 : Find equivalent states. The Table 8.4.3 shows the equivalent states in the given state table. The states A and C, and states B and E generate exactly same output and same next state. Hence, state A and C are equivalent and similarly states B and E are equivalent.

Present state	Next state, Z(output)	
	X = 0	X = 1
A	B, 0	C, 0
B	B, 0	D, 0
C	B, 0	C, 0
D	E, 1	C, 0
E	B, 0	D, 0

Equivalent states

Equivalent states

Table 8.4.3 Equivalent states

Step 2 : Replace redundant states with equivalent states.

∴ Replace C by A and replace E by B, and remove states C and E.

Now, there are no equivalent states and hence Table 8.4.3 (a) shows the minimized state table.

Present state	Next state, Z (output)	
	X = 0	X = 1
A	B, 0	A, 0
B	B, 0	D, 0
D	B, 1	A, 0

Table 8.4.3 (a) Minimized state table

Example 8.4.2 *Minimize the state table shown given below :* **Dec.-05, Marks 6**

Present state	Next state, Z (Output)	
	X (Input)	
	0	1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	G, 0	F, 1
G	A, 0	F, 1

Solution :

Step 1 : Find equivalent states

Present state	Next state, Z(output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	G, 0	F, 1
G	A, 0	F, 1

Equivalent states

Table 8.4.4 (a) Equivalent states

Step 2 : Replace redundant state with equivalent state.

Therefore, replace G by E and remove state G. Looking at Table 8.4.4 (b), we find that states D and F are equivalent. Therefore, replace F by D and remove state F. Now, there are no equivalent states and hence Table 8.4.4 (c) shows the minimized state table.

Present state	Next state, Z(output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	F, 1
E	A, 0	F, 1
F	E, 0	F, 1

Equivalent states

Table 8.4.4 (b) Equivalent states

Present state	Next state, Z (Output)	
	X = 0	X = 1
A	A, 0	B, 0
B	C, 0	D, 0
C	A, 0	D, 0
D	E, 0	D, 1
E	A, 0	D, 1

Table 8.4.4 (c) Minimized state table

Example 8.4.3 *i) Reduce the number of states in the following state table and tabulate the reduced state table.*

Present state	Next state		Output	
	x = 0	x = 1	x = 0	x = 1
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

ii) Starting from state a, and input sequence 01110010011, determine the output sequence for the given and reduced state table. **Dec.-07, 11, May-10, Marks 16**

Solution :

i)

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	f	b	0	0
b	d	c	0	0
c	f	e	0	0
d	g	a	1	0
e	d	c	0	0
f	f	b	1	1
g	g	h	0	1
h	g	a	1	0

Equivalent states (b, e)

Equivalent states (d, h)

Table 8.4.5

According to given state table we have states b and e, and states d and h are equivalent.

Present State	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	f	b	0	0
b	d	c	0	0
c	f	b	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

Equivalent states (b, c)

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	f	b	0	0
b	d	a	0	0
d	g	a	1	0
f	f	b	1	1
g	g	d	0	1

Reduced State Table

ii) Using reduced state table

Input sequence	State transition	Output
0	a → f	0
1	f → b	1
1	b → a	0
1	a → b	0
0	b → d	0
0	d → g	1
1	g → d	1
0	d → g	1
0	g → g	0
1	g → d	1
1	d → a	0

The output sequence for reduced state table is 01000111010.

Using given state table

Input sequence	State transition	Output
0	a → f	0
1	f → b	1
1	b → c	0
1	c → e	0
0	e → d	0
0	d → g	1
1	g → h	1
0	h → g	1
0	g → g	0
1	g → h	1
1	h → a	0

The output sequence for given state stable is also 01000111010.

8.4.2 Reduction of State Table using Partition Technique

A procedure for state minimization and determination of n-equivalence is discussed in the following example. Consider the same state table given in Table 9.2.

Step 1 : Partition the states into subsets such that all states in the same subsets are 1-equivalent.

The first partition P_1, can be obtained by placing those, states having the same outputs under all inputs, in the same block. This partitioning gives two subsets.

1. (A, C, E) : Their outputs are under 0 and 1 inputs are 0 and 1, respectively.
2. (B, D, F) : There outputs are under 0 and 1 inputs are 0 and 0, respectively.

$\therefore \quad P_1 = (A, C, E)\ (B, D, F)$

Step 2 : Partition the states into subsets such that all states in the same subsets are 2-equivalent.

This can be accomplished by observing that two states are 2-equivalent if and only if they are 1-equivalent and their I_i-successors, for all possible I_i, are also 1-equivalent. In other words we can say that, two states are placed in the same block of partition P_2 if and only if they are in the same block of P_1 and for each possible I_i there I_i-successor are also contained in a block of P_1.

1. The 0 and 1-successors of (A, C, E) are (C, E) and (B, D, F), respectively, and since both are contained in common blocks of P_1, the states (A, C, E) are 2-equivalent, and therefore (A, C, E) constitutes a block in P_2.
2. The 0-successor of (B, D, F) is (B, F) which contained in the common block of P_1; however 1-successor of (B, D, F) is (D, B, C) in which (D, B) and (C) are not contained in a single block of P_1. Therefore, the block (B, D, F) must be split into (BD) and (F).

$\therefore \quad P_2 = (A, C, E)\ (B, D)\ (F)$

Step 3 : Partition the states into subsets such that all states in the same subsets are 3-equivalent.

By applying same procedure in step 2, we can split block (A, C, E) of P_2 into (A, C) and (E), since the 1-successor of (A, C, E) is (D, B, F) which is not in the single block of P_2.

$\therefore \quad P_3 = (A, C)\ (E)\ (B, D)\ (F)$

Further partitioning of states is not possible because we find that the 0 and 1-successors of blocks in P_3 are in the common block of P_3.

The states in the common blocks of P_3 are equivalent and redundant. Therefore, state A = C and B = D. The states C and D are redundant states and can be eliminated to give minimised state table as shown in Table 8.4.6.

Present state	X = 0		X = 1	
	Next state	z	Next state	z
A	E	0	B	1
B	F	0	B	0
E	A	0	F	1
F	B	0	A	0

Table 8.4.6 Minimized state table

In general, we can say that, the P_{k+1} partition is obtained from P_K by placing in the same block of P_{K+1} those states which are in the same block of P_K and whose I_i-successors for every possible I_i are also in a common block of P_K.

When $P_{K+1} = P_K$, the partitioning process terminates and P_K defines the sets of equivalent states of the sequential machine. The P_K is thus called the **equivalence partition** and the partitioning procedure discussed above is referred to as the **Moore reduction procedure**.

Example 8.4.4 *Determine the minimal state equivalent of the state table given :*

PS	NS, Z	
	x = 0	x = 1
q_0	$q_0, 1$	$q_4, 0$
q_1	$q_0, 0$	$q_4, 0$
q_2	$q_1, 0$	$q_5, 0$
q_3	$q_1, 0$	$q_5, 0$
q_4	$q_2, 0$	$q_6, 1$
q_5	$q_2, 0$	$q_6, 1$
q_6	$q_3, 0$	$q_7, 1$
q_7	$q_3, 0$	$q_7, 1$

Solution :

1. States having the same outputs, under all input conditions can be grouped as
   $P_1 : (q_0)\ (q_1, q_2, q_3)\ (q_4, q_5, q_6, q_7)$
2. 1-successors of (q_1, q_2, q_3) are in two different blocks and therefore,
   $P_2 : (q_0)\ (q_1)\ (q_2, q_3)\ (q_4, q_5, q_6, q_7)$
3. 1-successor of (q_2, q_3) and (q_4, q_5, q_6, q_7) are in the common blocks of P_2.

   Therefore,
   $P_3 : (q_0)\ (q_1)\ (q_2, q_3)\ (q_4, q_5, q_6, q_7)$
   Thus equivalent states are :
   $q_2 = q_3$ and $q_4 = q_5 = q_6 = q_7$

The resultant minimized state table is as shown in Table 8.5.7.

PS	NS, Z	
	X = 0	X = 1
q_0	q_0, 1	q_4, 0
q_1	q_0, 0	q_4, 0
q_2	q_1, 0	q_4, 0
q_4	q_2, 0	q_4, 1

Table 8.4.7 Minimized state table

Example 8.4.5 *Determine a minimal state table equivalent to the state table given :*

PS	NS, Z	
	X = 0	X = 1
1	1, 0	1, 0
2	1, 1	6, 1
3	4, 0	5, 0
4	1, 1	7, 0
5	2, 0	3, 0
6	4, 0	5, 0
7	2, 0	3, 0

Solution :

1. States having the same outputs, under all input conditions can be grouped as
   P_1 : (1, 3, 5, 6, 7) (2) (4)
2. 1-successors of (1, 3, 5, 6, 7) are (1, 4, 2, 4, 2) when X = 0 and (1, 5, 3, 5, 3) when X = 1, which are in different blocks and therefore,
   P_2 : (1) (3, 6) (5, 7) (2) (4)
3. 1-successors of (3, 6) and (5, 7) are in the common block of P_2. Therefore,
   P_3 : (1) (3, 6) (5, 7) (2) (4)

Thus equivalent states are : 3 = 6 and 5 = 7

The resultant minimized state table is as shown in Table. 8.4.8.

PS	NS, Z	
	x = 0	x = 1
1	1, 0	1, 0
2	1, 1	3, 1
3	4, 0	5, 0
4	1, 1	5, 0
5	2, 0	3, 0

Table 8.4.8 Minimized state table

8.4.3 Reduction of State Table using Merger Graph

Merger graphs is state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows :

1. It contains the same number of vertices as the state table contains states. Refer Fig. 8.4.3.
2. Each compatible state pair is indicated by a line drawn between the two state vertices.
3. Every potentially compatible state pair, with outputs not in conflict but whose next states are different, is connected by a broken line. The implied states are drawn in the line break between the two potentially compatible states.
4. If two states are incompatible, no connecting line is drawn.

Example 8.4.6 *Reduce the state table shown in Table 8.4.9 using merger graph method.*

Present state	Next state , XY				Output Z
	00	01	11	10	
A	A	B	–	C	0
B	A	B	D	–	0
C	A	–	D	C	0
D	–	B	D	E	0
E	A	–	D	E	1

Table 8.4.9 State table

Solution :

1. States A and B are compatible. Thus the line is drawn between A and B.
2. State A and C are compatible. Thus the line is drawn between A and C.
3. State B and C are compatible. Thus, the line is drawn between B and C.
4. States A and D are compatible only if implied states C and E are compatible. This is indicated by drawing a broken line between A and D with CE written in between.

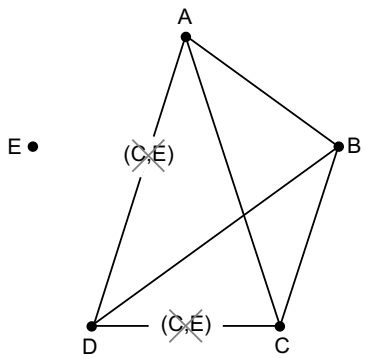


Fig. 8.4.3

5. States A and E are incompatible since there outputs are different, so line is not drawn between A and E. For the same reason states B, C and D are also not compatible with E.
6. State B and D are compatible. Thus, the line is drawn between B and D.
7. States C and D are compatible only if implied states C and E are compatible. This is indicated by drawing a broken line between C and D with CE written in between.

8. It is found that states C and E are not compatible and hence states A and D, and states D and C are also not compatible. This is indicated by cross (×) marks.

The compatibility lines of merger graph form a geometrical pattern consisting of one triangle (A, B, C), a line (B, D) and a single state E that is not compatible with any others. Thus maximum compatibles are :

(A, B, C) (B, D) (E)

Here, we can notice that state B is common in two sets. However, it can be compatible with either states A and C or state D, but not both. If we consider the next don't care state of B as state C, it is compatible with stable A and C. If we consider the next don't care state of B as state E it is compatible with state D.

Considering state B compatible with states A and C, we have following set of maximum compatibilities for given state table

(A, B, C) (D) (E)

Considering state B compatible with state D, we have following set of maximum compatibilities for given state table.

(A, C) (B, D) (E)

So we can say that there may be more than one possible way of merging rows when reducing a state table.

According to alternative-I we have

$(A, B, C) \rightarrow S_0$

$(D) \rightarrow S_1$

$(E) \rightarrow S_2$

and according to alternative-II we have

$(A, C) \rightarrow S_0$

$(B, D) \rightarrow S_1$

$(E) \rightarrow S_2$

The Table 8.4.9 (a) shows the reduced state table using both the alternatives.

Present state	Next state , XY				Output Z
	00	01	11	10	
S_0	S_0	S_0	S_1	S_0	0
S_1	–	S_0	S_1	S_2	0
S_2	S_0	–	S_1	S_2	1

Reduced state table (Alternative I)

Present state	Next state , XY				Output Z
	00	01	11	10	
S_0	S_0	S_1	S_1	S_0	0
S_1	S_0	S_1	S_1	S_2	0
S_2	S_0	–	S_1	S_2	1

Reduced state table (Alternative II)

Table 8.4.9 (a)

Example 8.4.7 *Reduce the given state table.*

Present state	Next state, for inputs SR/output Q			
	00	**01**	**11**	**10**
A	A/0	C/0	–	B/0
B	D/1	–	E/1	B/1
C	A/0	C/0	F/0	–
D	D/1	C/1	–	B/1
E	–	C/0	E/0	G/0
F	–	H/1	F/1	B/1
G	A/0	–	E/0	G/0
H	D/1	H/1	F/1	–

Table 8.4.10

Solution : The Fig. 8.4.4 shows the merger graph. Each compatible state pair is indicated by a line drawn between the two states vector.

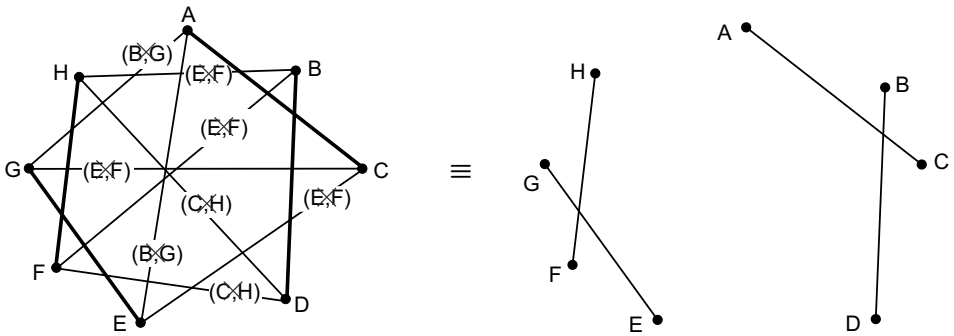


Fig. 8.4.4 Merger graph for state table

Present state	Next State, for inputs SR/output Q			
	00	**01**	**11**	**10**
A	A/0	C/0	**F/0**	B/0
B	D/1	**C/1**	E/1	B/1
C	A/0	C/0	F/0	**B/0**
D	D/1	C/1	**E/1**	B/1
E	**A/0**	C/0	E/0	G/0
F	**D/1**	H/1	F/1	B/1
G	A/0	**C/0**	E/0	G/0
H	D/1	H/1	F/1	**B/1**

Equivalent states (S_1) — B, D

Equivalent states (S_3) — F, H

Equivalent states (S_0) — A, C

Equivalent states (S_2) — E, G

Table 8.4.11

Every potentially compatible state pair, with outputs not in conflict but whose next states are different, is connected by a broken line. The implied states are drawn in the line break between the two potentially compatible states. If two states are incompatible, no connecting line is drawn.

Therefore, we have

$(A, C) \rightarrow S_0$

$(B, D) \rightarrow S_1$

$(E, G) \rightarrow S_2$

$(F, H) \rightarrow S_3$

The Table 8.4.12 shows the reduced primitive flow.

Present state	Next state for inputs SR/output Q			
	00	01	11	10
S_0	S_0 , 0	S_0 , 0	S_3, 0	S_1, 0
S_1	S_1 , 1	S_0, 1	S_2, 1	S_1 , 1
S_2	S_0, 0	S_0, 0	S_2 , 0	S_2 , 0
S_3	S_1, 1	S_3 , 1	S_3 , 1	S_1, 1

Table 8.4.12 Reduced state table

8.4.4 Reduction of State Table using Merger Table

When performing state reduction of machines having a large number of states, it may be more convenient to find the compatible pair and their implications in a merger table of the form illustrated in Fig. 8.4.5, instead of using the merger graph. The merger table method is also called as **Paull-Unger method or implication chart method.** The table shows the state table for sequential machine and its merger table in the Fig. 8.4.5.

PS	NS, z	
	I_1	I_2
A	E, 0	B, 0
B	F, 0	A, 0
C	E, –	C, 0
D	F, 1	D, 0
E	C, 1	C, 0
F	D, –	B, 0

Table 8.4.13 State table

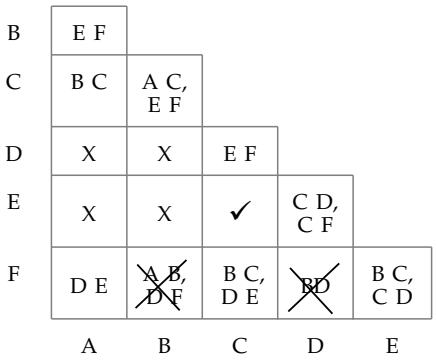


Fig. 8.4.5 Merger table

Each cell of the merger table corresponds to the compatible pair represented by the intersection of the row and column headings. The incompatibility of two states is indicated by placing cross mark X in the corresponding cell, while their compatibility is indicated by placing check mark (✓) in the cell. For example, states A and D are incompatible because their outputs are conflicting and hence the cell corresponding them

contain cross mark X. On the other hand, states C and E are compatible and hence the cell corresponding them contain check mark ✓. The implied pair or pairs corresponding to state-pair are written with the cell, as shown in the Fig. 8.4.5 For example, states A and C are compatible only when implied states B and C are compatible, therefore implied pair B C is written in the cell corresponding to states A and C. In the similar way the entire merger table is written. Now it is necessary check whether implied pairs are compatible or not. This can be checked by observing the merger table. The implied states are incompatible if corresponding cell contain X. For example, implied pair BD is incompatible because cell BD contain X. Since BD is incompatible, states D and F are also incompatible. As DF is compatible, states B and F are also incompatible. It is indicated by cross mark in Fig. 8.4.5.

Once the merger table is completed, the set of all maximal compatibles can be found by procedure which is counter part to that of finding complete polygon in the merger graph. The procedure is as follows :

1. Begin with right most column in the merger table and proceed left unit a column containing a compatible pair is encountered. For example, the pair EF in the merger table shown in Fig. 8.4.5.
2. Proceed left to the next column containing at least one compatible pair. If the state to which this column corresponds is compatible with all the states in the set of previously determined compatible states then add this state to that set of compatible states to form a larger compatible. If the state is not compatible with all the states of previously determined set, but is compatible with some other state, form a new set of compatible states. For example, state D is not compatible with both E and F which are previously determined set of states. However, it is compatible with state E. Therefore, it is not added in the previous set, but the new set is formed, DE. On the other hand, state C is compatible with both E and F which are previously determined set of states and hence it is included in that set to form a new set CEF. Similarly, C is also included in set DE to get a new set CDE.
3. Repeat step 2 until the left most column is reached.

   After application of above mentioned procedure the merger table in Fig. 8.4.5, gives following set of maximal compatibles.

   Column E : (E, F)

   Column D : (E, F); (D, E)

   Column C : (C, E, F); (C, D, E)

   Column B : (C, E, F); (C, D, E); (B, C)

   Column A : (C, E, F); (C, D, E); (A, B, C); (A, C, F)

The right most column is E. It indicates that the state E is compatible with state F, resulting a compatible pair (EF). The column D indicates that the state D is compatible with state E, thus compatible pair (DE) is included in the compatible list. The column C shows that the state C is compatible with states D, E and F and consequently the compatibles generated previously are enlarged to include state C. The column B consists of a single compatible pair (BC), which is added to the previously generated compatible list. The column A indicates that the state A is compatible with states B, C and F. Thus, the previously generated compatible pair (BC) is enlarged to give the compatible (ABC), while rows C and F, together with preciously available compatibility relations, give the compatible (ACF). Therefore, the set of maximal compatibles is

(C, E, F); (C, D, E); (A, B, C); (A, C, F)

8.4.5 State Assignment

In sequential circuits we know that the behaviour of the circuit is defined in terms of its inputs, present states, next states and outputs. To generate desired next state at particular present state and inputs, it is necessary to have specific flip-flop inputs. These flip-flop inputs are described by a set of Boolean functions called flip-flop input functions. To determine the flip-flop input functions, it is necessary to represent states in the state diagram using binary values instead of alphabets. This procedure is known as state assignment. We must assign binary values to the states in such a way that it is possible to implement flip-flop input functions using minimum logic gates. Fig. 8.4.6 shows the state diagram shown in Fig. 8.4.2 after state assignments.

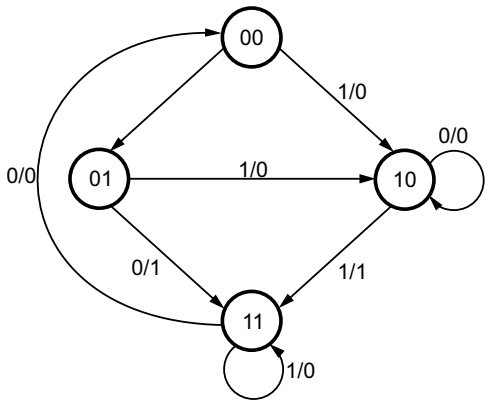


Fig. 8.4.6 State diagram with binary states

Rules for state assignments

There are two basic rules for making state assignments.

Rule 1 : States having the **same** NEXT STATES for a given input condition should have assignments which can be grouped into logically adjacent cells in a K-map.

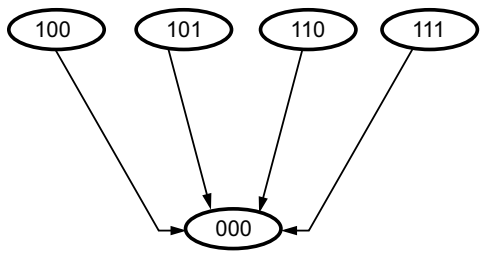


Fig. 8.4.7 Example of using Rule 1

Fig. 8.4.7 shows the example for Rule 1. As shown in the Fig. 8.4.7, there are four states whose next state is same. Thus states

assignments for these states are 100, 101,110 and 111 which can be grouped into logically adjacent cells in a K-map.

Rule 2 : States that are the NEXT STATES of a single state should have assignment which can be grouped into logically adjacent cells in a K-map.

Fig. 8.4.8 shows the example for Rule 2. As shown in the Fig. 8.4.8, for state 000, there are four next states. These states are assigned as 100, 101, 110 and 111 so that they can be grouped into logically adjacent cells in a K-map and table shows the state table with assigned states.

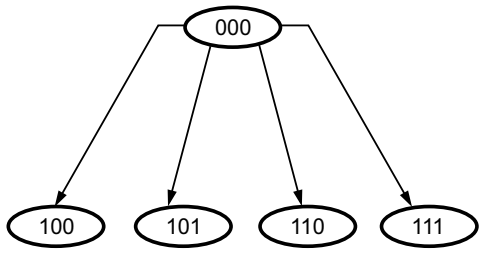


Fig. 8.4.8 Example of using Rule 2

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
00	01	10	0	0
01	11	10	1	0
10	10	11	0	1
11	00	11	0	0

Table 8.4.14 State table with assigned states

8.4.6 Choice of Flip-Flops and Derivation of Next State and Output

From the state assigned state table shown in Table 8.4.14, we can derive the logic expression for the next state and output functions. But first we have to decide on the type of flip-flops that will be used in the circuit. The most straightforward choice is to use D flip-flops, because in this case the values of next state are simply clocked into the flip-flops to become the new values of present state. For other types of flip-flops, such as JK, T and RS the relationship between the next-state variable and inputs to a flip-flop is not as straightforward as D flip-flop. For other types of flip-flops we have to refer excitation table of flip-flop to find flip-flop inputs. This is illustrated in the following example.

Example 8.4.8 *A sequential circuit has one input and one output. The state diagram is shown in Fig. 8.4.9. Design the sequential circuit with a) D flip-flops b) T flip-flops c) RS flip-flops and d) JK- flip-flops.*

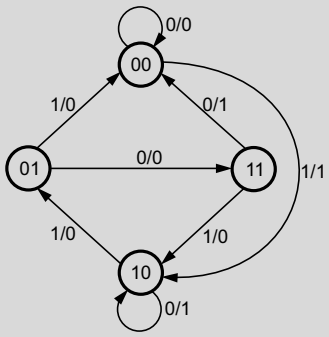


Fig. 8.4.9

Solution : The state table for the state diagram shown in Fig. 8.4.9 is as given in Table 8.4.15.

Present state		Next state		Output	
		X = 0	X = 1	X = 0	X = 1
A	B	AB	AB	Y	Y
0	0	0 0	1 0	0	1
0	1	1 1	0 0	0	0
1	0	1 0	0 1	1	0
1	1	0 0	1 0	1	0

Table 8.4.15

As seen from the state table there are no equivalent states. Therefore, no reduction is the state diagram. The state table shows that circuit goes through four states, therefore we require 2 flip-flops (number of states = 2^m, where m = number of flip-flops). Since two flip-flops are required first is denoted as A and second is denoted as B.

1. Design using D Flip-Flops

As mentioned earlier, for D flip-flops next states are nothing but the new present states. Thus, we can directly use next states to determine the flip-flop input with the help of K-map simplification.

K-map simplification

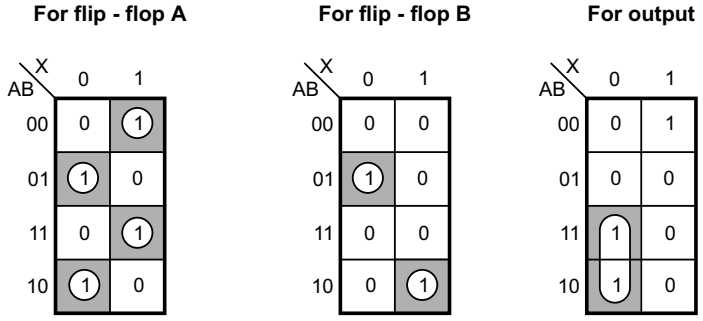


Fig. 8.4.10

$$D_A = \overline{A}\,\overline{B}\,X + \overline{A}\,B\,\overline{X} + ABX + A\,\overline{B}\,\overline{X} \quad \text{and}$$

$$D_B = \overline{A}\,B\,\overline{X} + A\,\overline{B}\,X$$

$$Y = \overline{A}\,\overline{B}\,X + A\,\overline{X}$$

With these flip-flop input functions and circuit output function we can draw the logic diagram as follows

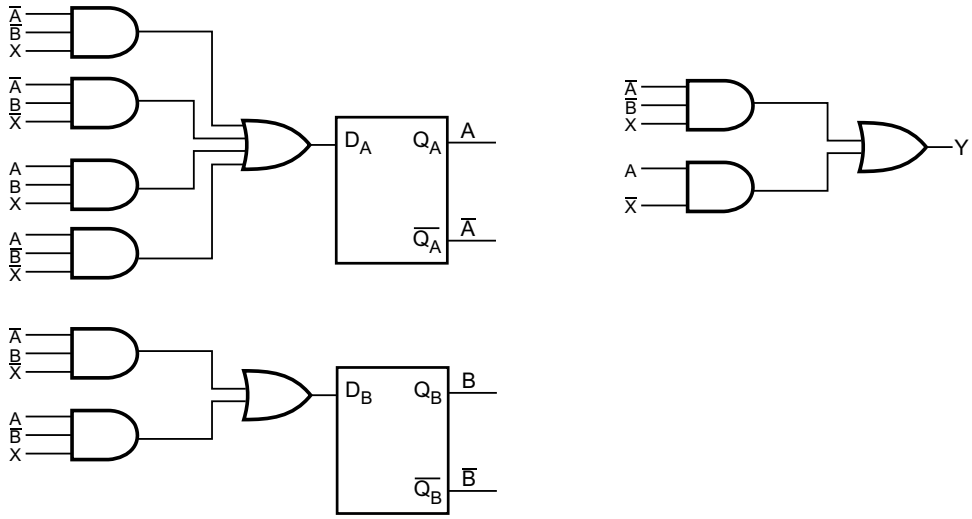


Fig. 8.4.11 Logic diagram of given sequential circuit using D flip-flop

2. Design using T flip-flops

Using the excitation table for T flip-flop shown in Table 8.4.16 we can determine the excitation table for the given circuit as shown in Table 8.4.17.

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Table 8.4.16 Excitation table for T flip-flop

Present state		Input	Next state		Flip-flop inputs		Output
A	B	X	A	B	T_A	T_B	Y
0	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1
0	1	0	1	1	1	0	0
0	1	1	0	0	0	1	0
1	0	0	1	0	0	0	1
1	0	1	0	1	1	1	0
1	1	0	0	0	1	1	1
1	1	1	1	0	0	1	0

Table 8.4.17 Circuit excitation table

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from 0 → 0 for T flip-flop requires input T to be at logic 0. The second row shows that flip-flop A has transition 0 → 1. It requires the input T_A to be at logic 1. It requires the input T_A to be at logic 1. Similarly, we can find inputs for each flip-flop for each row in the table by referring present state, next state and excitation table.

Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

K-map simplification

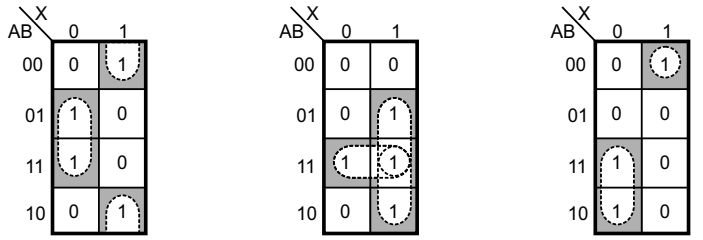


Fig. 8.4.12

Therefore, input function for $T_A = B\overline{X} + \overline{B}X$,

$T_B = AB + BX + AX$, and

Circuit output function $= A\overline{X} + \overline{A}\,\overline{B}X$

With these flip-flop input functions and circuit output function we can draw the logic diagram as follows :

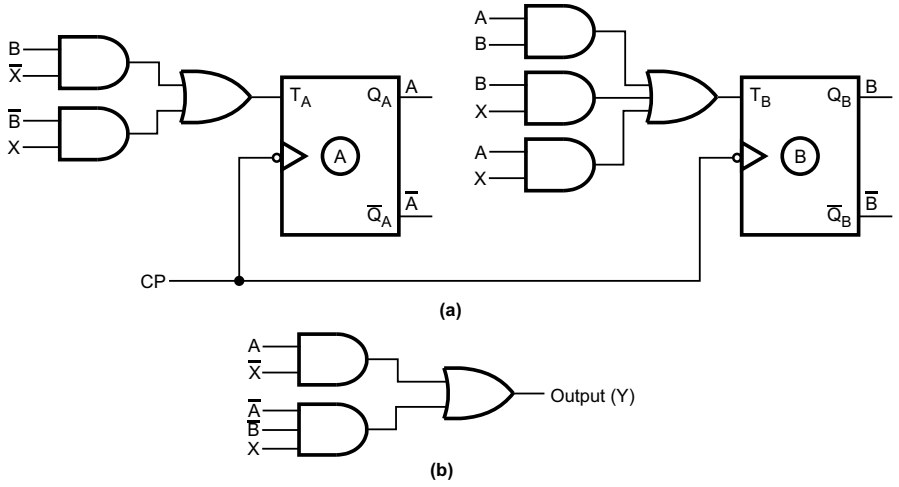


Fig. 8.4.13 Logic diagram of given sequential circuit using T flip-flops

3. Design using RS Flip-Flops

Using the excitation table for RS flip-flop shown in Table 8.4.18 we can determine the excitation table for the given circuit as shown in Table 8.4.19.

Q_n	Q_{n+1}	R	S
0	0	X	0
0	1	0	1
1	0	1	0
1	1	0	X

Table 8.4.18 Excitation table for RS flip-flop

Present state		Input	Next state		Flip-flop inputs				Output
A	B	X	A	B	R_A	S_A	R_B	S_B	Y
0	0	0	0	0	X	0	X	0	0
0	0	1	1	0	0	1	X	0	1
0	1	0	1	1	0	1	0	X	0
0	1	1	0	0	X	0	1	0	0
1	0	0	1	0	0	X	X	0	1
1	0	1	0	1	1	0	0	1	0
1	1	0	0	0	1	0	1	0	1
1	1	1	1	0	0	X	1	0	0

Table 8.4.19

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from $0 \rightarrow 0$ for RS flip-flop requires inputs R and S to be X and 0, respectively. Similarly, we can determine inputs for each flip-flop for each row in the table by referring present state, next state and excitation table. Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

K-map simplification

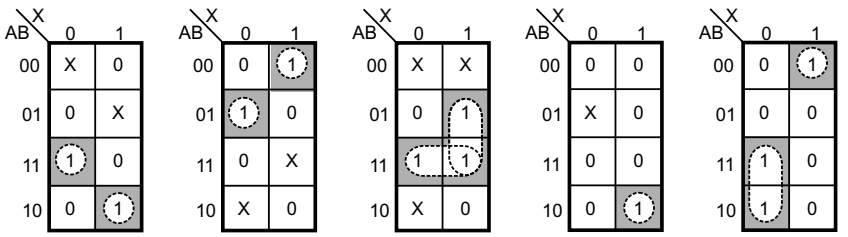


Fig. 8.4.14

Therefore input function for

$$R_A = AB\overline{X} + A\overline{B}X$$

$$S_A = \overline{A}\,B\overline{X} + \overline{A}\,\overline{B}X$$

$$R_B = AB + BX$$

$$S_B = A\overline{B}\,X \text{ and,}$$

$$\text{Circuit output function} = A\overline{X} + \overline{A}\,\overline{B}\,X$$

With these flip-flop input functions and circuit output function we can draw the logic diagram as follows :

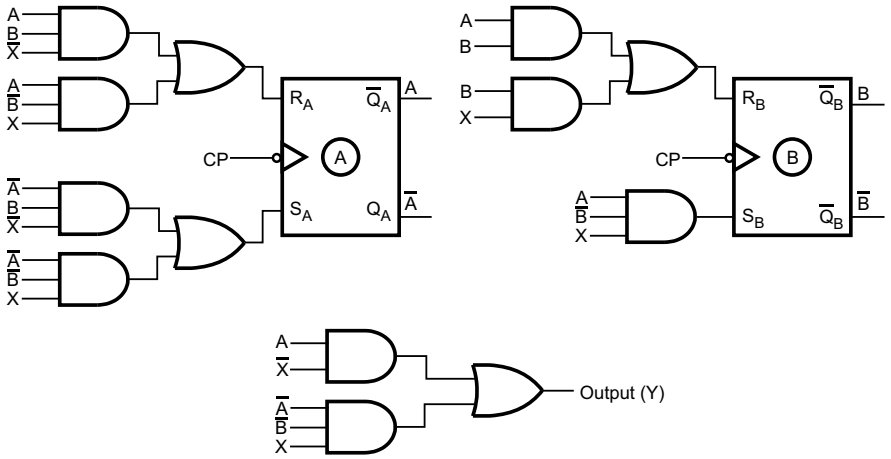


Fig. 8.4.15 Logic diagram of given sequential circuit using RS flip-flop

4. Design using JK Flip-Flops

Using the excitation table for JK flop-flop shown in Table 8.4.20 we can determine the excitation table for the given circuit as shown in Table 8.4.21.

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Table 8.4.20 Excitation table for JK flip-flop

Present state		Input	Next state		Flip-flop inputs				Output
A	B	X	A	B	J_A	K_A	J_B	K_B	Y
0	0	0	0	0	0	X	0	X	0
0	0	1	1	0	1	X	0	X	1
0	1	0	1	1	1	X	X	0	0
0	1	1	0	0	0	X	X	1	0
1	0	0	1	0	X	0	0	X	1
1	0	1	0	1	X	1	1	X	0
1	1	0	0	0	X	1	X	1	1
1	1	1	1	0	X	0	X	1	0

Table 8.4.21

The first row of circuit excitation table shows that there is no change in the state for both flip-flops. The transition from $0 \rightarrow 0$ for JK flip-flop requires inputs J and K to be 0 and X, respectively. Similarly, we can determine inputs for each flip-flop for each row in the table by referring present state, next state and excitation table. Let us use K-map simplification to determine the flip-flop input functions and circuit output functions.

K-map simplification

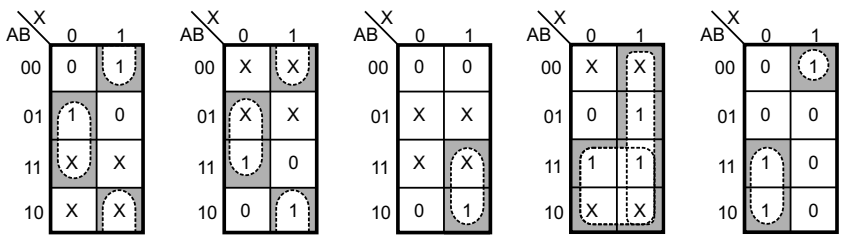


Fig. 8.4.16

Therefore, input function for

$$J_A = B\overline{X} + \overline{B}X$$

$$K_A = B\overline{X} + \overline{B}X$$

$$J_B = AX$$

$$K_B = A+X$$

$$\text{Circuit output function} = A\overline{X} + \overline{A}\,\overline{B}X$$

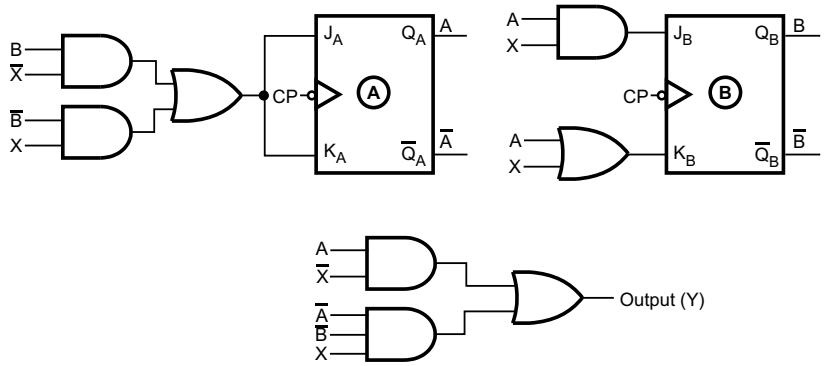


Fig. 8.4.17 Logic diagram of given sequential circuit using JK flip-flop

8.4.7 State Assignment Problem

Example 8.4.9 *Design a sequential circuit for a state diagram shown in Fig. 8.4.18.*

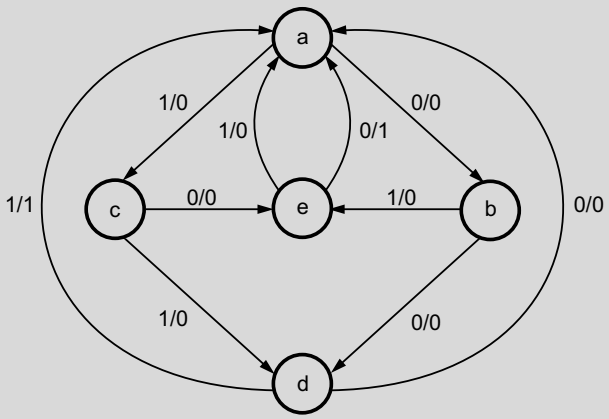


Fig. 8.4.18

Solution :

Step 1 : Assign binary values to each state. Using random state assignment we get

a = 000, b = 001, c = 010, d = 011 and e =100. The excitation table with these assignments is as given in Table 8.4.22.

Step 2 : Find number of flip-flops needed. Since there are five states we require $2^n \geq 5$ $\therefore$ n = 3 flip-flops.

Let us use D flip-flops

Step 3 : Derive the excitation table.

Present state			Input	Next state			Output
A_n	B_n	C_n	X	A_{n+1}	B_{n+1}	C_{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	0
0	1	0	1	0	1	1	0
0	1	1	0	0	0	0	0
0	1	1	1	0	0	0	1
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Table 8.4.22

Step 4 : Derive circuit output and flip-flop input functions using K-map simplification

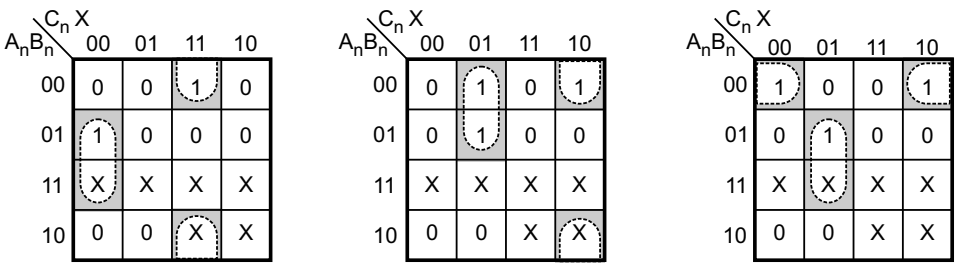

C_n X
A_nB_n
00 01 11 10
00 0 0 1 0
01 1 0 0 0
11 X X X X
10 0 0 X X
$D_A = B_n\overline{C}_n\overline{X} + \overline{B}_nC_nX$
C_n X
A_nB_n
00 01 11 10
00 0 1 0 1
01 0 1 0 0
11 X X X X
10 0 0 X X
$D_B = \overline{A}_n\overline{C}_nX + \overline{B}_nC_n\overline{X}$
C_n X
A_nB_n
00 01 11 10
00 1 0 0 1
01 0 1 0 0
11 X X X X
10 0 0 X X
$D_C = \overline{A}_n\overline{B}_n\overline{X} + B_n\overline{C}_nX$

Fig. 8.4.19 (a)

The random assignments require:

7 three input AND functions
1 two input AND function
4 two input OR functions
12 gates with 31 inputs

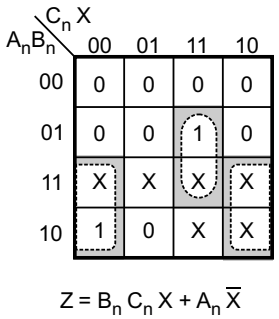


Fig. 8.4.19 (b)

Step 5 : State assignment using rules.

Now we will apply the state assignment rules and compare the results.

Looking at Fig. 8.4.19 (a) and (b).

Rule 1 says that : e and d must be adjacent, and
b and c must be adjacent

Rule 2 says that : b and c must be adjacent, and
e and d must be adjacent

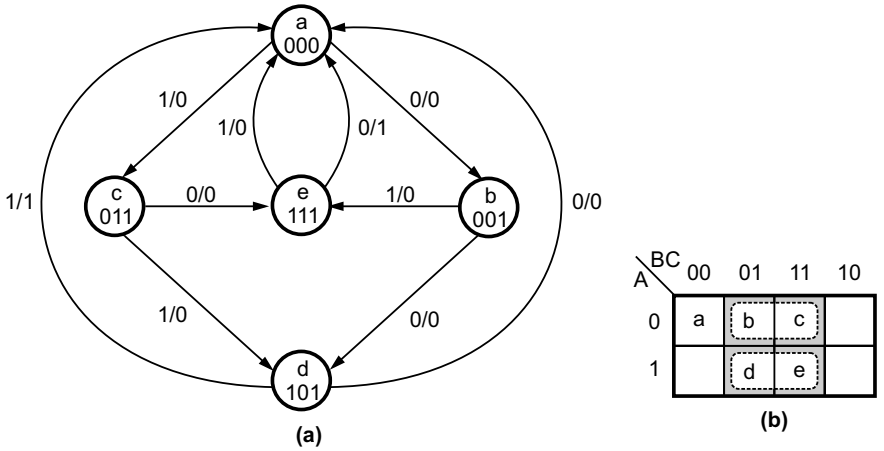


Fig. 8.4.20 (a) State diagram after applying Rules 1 and 2
(b) K-map showing b-c d-e adjacent to each other

Step 6 : Derive excitation table

Applying Rule 1 Rule 2 to the state diagram we get the excitation table as shown in the Table 8.4.23.

Present state			Input	Next state			Output
A_n	B_n	C_n	X	A_{n+1}	B_{n+1}	C_{n+1}	Z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	0
0	0	1	0	1	0	1	0
0	0	1	1	1	1	1	0
0	1	0	0	X	X	X	X
0	1	0	1	X	X	X	X
0	1	1	0	1	1	1	0
0	1	1	1	1	0	1	0
1	0	0	0	X	X	X	X
1	0	0	1	X	X	X	X
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

Table 8.4.23

Step 7 : Derive circuit output and flip-flop input functions using K-map simplification.

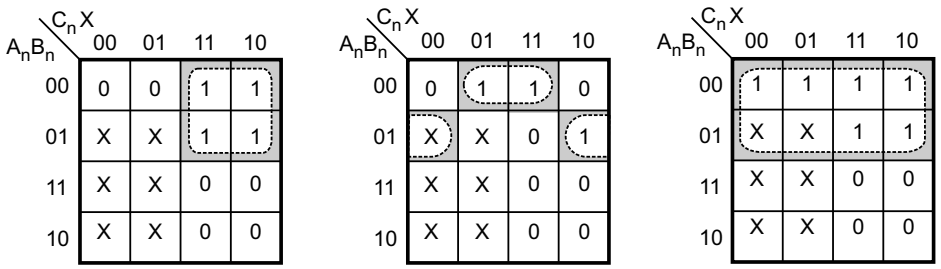

$C_n X$
A_nB_n
00 01 11 10
00 0 0 1 1
01 X X 1 1
11 X X 0 0
10 X X 0 0
$A_{n+1} = D_A = \overline{A}_n C_n$
$C_n X$
A_nB_n
00 01 11 10
00 0 1 1 0
01 X X 0 1
11 X X 0 0
10 X X 0 0
$B_{n+1} = D_B = \overline{A}_n\overline{B}_n X + \overline{A}_n\overline{B}_n\overline{X}$
$C_n X$
A_nB_n
00 01 11 10
00 1 1 1 1
01 X X 1 1
11 X X 0 0
10 X X 0 0
$C_{n+1} = D_C = \overline{A}_n$

Fig. 8.4.21 (a)

A_nB_n \ C_nX	00	01	11	10
00	0	0	0	0
01	X	X	0	0
11	X	X	0	1
10	X	X	1	0

$$Z = A_nB_n\overline{X} + A_nB_n\overline{X}$$

Fig. 8.4.21 (b)

The state assignments using Rule 1 and 2 require :

4 three input AND functions

1 two input AND functions

2 two input OR functions

7 gates with 18 inputs

Thus by simply applying Rules 1 and 2 good results have been achieved.

8.4.8 Design with Unused States

There are occasions when a sequential circuit may use less than the available these maximum number of states. We can consider the unused states as don't care conditions and can be used to simplify the input functions to flip-flops.

Let us consider one example. First we will design the given sequential circuit without using unused states and then we will design the given sequential circuit using unused states.

Example 8.4.10 *Design the sequential circuit for the state diagram shown in Fig. 8.4.22 use JK flip-flops.*

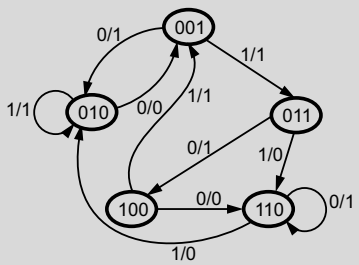


Fig. 8.4.22

Solution :

Step 1 : Derive excitation table

The excitation table for given state diagram is as follows

Present state			Input	Next state			Flip-flop inputs						Output
A	B	C	X	A	B	C	J_A	K_A	J_B	K_B	J_C	K_C	Y
0	0	1	0	0	1	0	0	X	1	X	X	1	1
0	0	1	1	0	1	1	0	X	1	X	X	0	1
0	1	0	0	0	0	1	0	X	X	1	1	X	0
0	1	0	1	0	1	0	0	X	X	0	0	X	1
0	1	1	0	1	0	0	1	X	X	1	X	1	1
0	1	1	1	1	1	0	1	X	X	0	X	1	0
1	0	0	0	1	1	0	X	0	1	X	0	X	0
1	0	0	1	0	0	1	X	1	0	X	1	X	1
1	1	0	0	1	1	0	X	0	X	0	0	X	1
1	1	0	1	0	1	0	X	1	X	0	0	X	0

Table 8.4.24

Step 2 : K-map simplification for JK inputs and circuit output

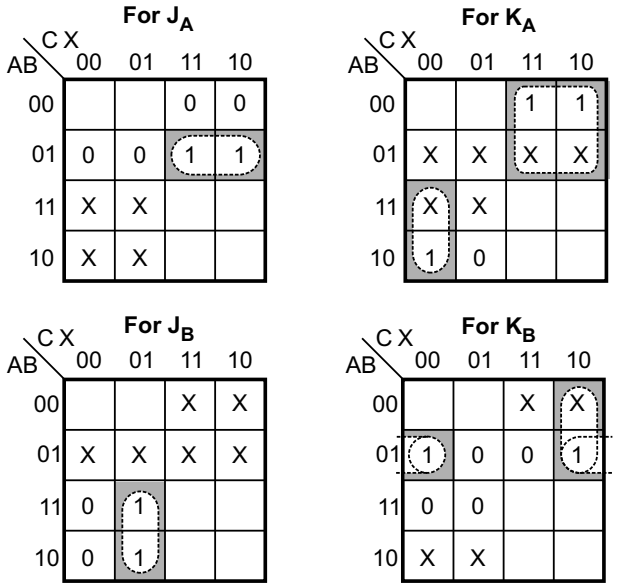


Fig. 8.4.23 (a)

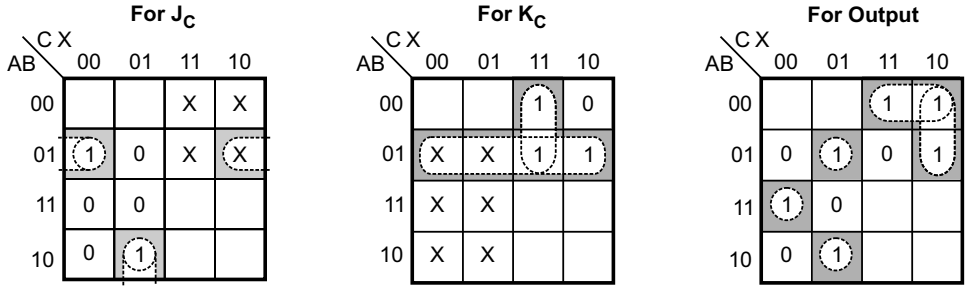


Fig. 8.4.23 (b)

Therefore, input functions for

$$J_A = \overline{A}BC$$

$$K_A = A\overline{C}\,\overline{X} + \overline{A}C$$

$$J_B = A\overline{C}X$$

$$K_B = \overline{A}B\,\overline{X} + \overline{A}C\overline{X}$$

$$J_C = \overline{A}B\,\overline{X} + A\overline{B}\,\overline{C}X$$

$$K_C = \overline{A}B + \overline{A}CX \text{ and}$$

Circuit output function, $Y = AB\,\overline{C}\,\overline{X} + \overline{A}B\overline{C} + A\overline{B}\,\overline{C}X + \overline{A}\,\overline{B}C + \overline{A}C\overline{X}$

Step 3 : Draw logic diagram.

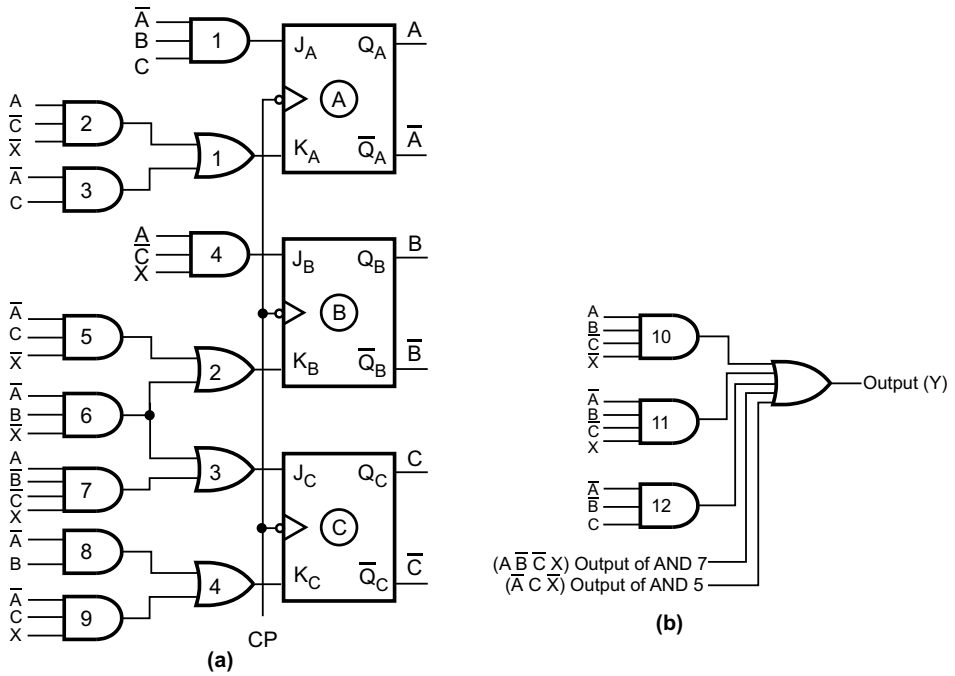


Fig. 8.4.24

Step 4 : Derive circuit output and flip-flop inputs considering unused states.

Let us see the circuit design with the use of unused states. These unused states 000, 101 and 111 are considered as a don't cares and are used to simplify the K-maps as follows :

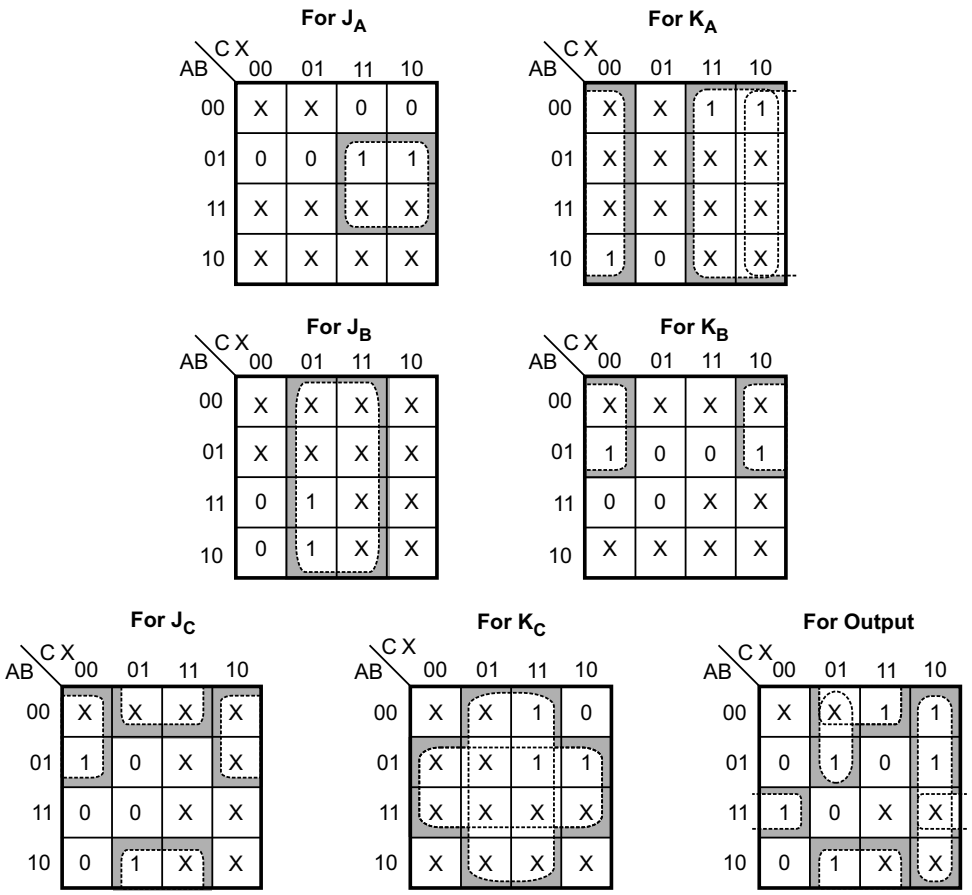


Fig. 8.4.25

Therefore, input functions for

$$J_A = BC$$

$$K_A = \overline{X} + C$$

$$J_B = X$$

$$K_B = \overline{A}\,\overline{X}$$

$$J_C = \overline{A}\,\overline{X} + \overline{B}X \quad \text{and} \quad K_C = B + X$$

The circuit output function $Y = AB\overline{X} + A\overline{C}X + \overline{B}X + C\overline{X}$

Step 5 : Draw logic diagram.

From the above logic diagram it can be realized that using unused state as don't cares we can further simplify the flip-flop input functions and circuit output function.

One important thing neglected up to this point in the design is the initial state of a sequential circuit. When power is turned 'ON', one cannot predict the initial state of the flip-flops. To initialize the states of all flip-flops to a specific states, preset and clear inputs of the flip-flops are used. These signals are activated accordingly before the clocked operation start. This procedure ensures the valid initial state. However, because of a noise signal or any other unforeseen reason, if the circuit goes in an invalid state, it is necessary to ensure that the circuit eventually goes into one of the valid states so it can resume normal operation.

Let us analyze the previous example to see whether the next state from the unused state is valid state or not. We know that unused states are 000, 101 and 111. These states are included in the state diagram and transitions are shown using input functions derived from the K-maps.

From the state diagram shown in Fig. 8.4.27 it is clear that the next state of the unused state is a valid state for all inputs.

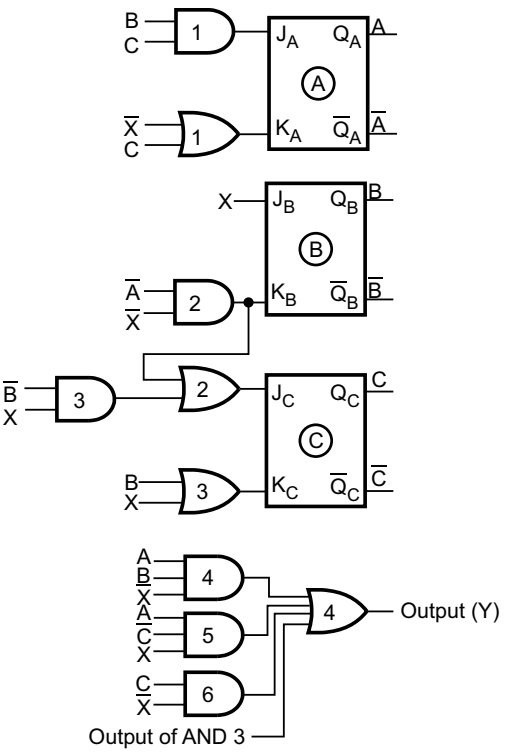


Fig. 8.4.26 Logic diagram

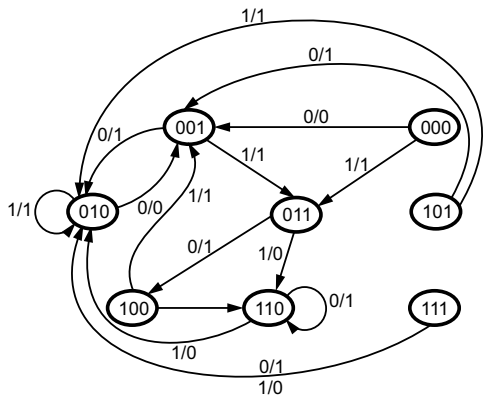


Fig. 8.4.27 State diagram

8.4.9 Lockout Condition

In a counter if the next state of some unused state is again an unused state and if by chance the counter happens to find itself in the unused states and never arrived at a used state then the counter is said to be in the lockout conditions. This is illustrated in the Fig. 8.4.28. The counter which never goes in lockout condition is called **self starting counter.**

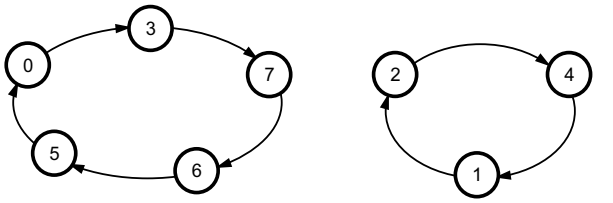


Fig. 8.4.28

The circuit that goes in lockout condition is called **bushless circuit**. To make sure that the counter will come to the initial state from any unused state, the additional logic circuit is necessary. To ensure that the lock out does not occur, the counter should be designed by forcing the next state to be the initial state from the unused states as shown in Fig. 8.4.29.

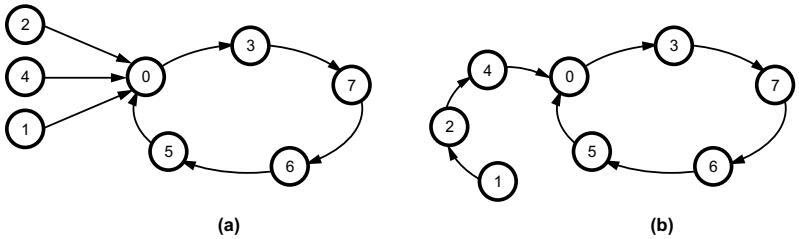


Fig. 8.4.29 State diagram for removing lockout

For example, as shown in Fig. 8.4.29, actually it is not necessary to force all unused states into initial state. Forcing any one state is sufficient. Because if counter initially goes to unused state which is not forced, it will go to another unused state. This will continue until it reaches the forced unused state. Once forced unused state is reached next state is used state, and circuit is lock free circuit. This is illustrated in Fig. 8.4.29 (b).

Example 8.4.11 *Design a synchronous counter for*
$4 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 4 \ldots$
Avoid lockout condition. Use JK type design.

Solution :

Step 1 : State diagram

Here, states 5, 2 and 0 are forced to go into 6, 3 and 1 state, respectively to avoid lockout condition.

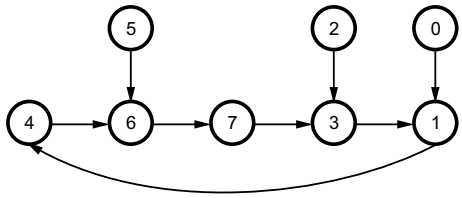


Fig. 8.4.30

Step 2 : Excitation table

Present states			Next states			Flip-flop inputs					
A	B	C	A_{+1}	B_{+1}	C_{+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	1	0	0	1	X	0	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	0	0	1	0	X	X	1	X	0
1	0	0	1	1	0	X	0	1	X	0	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	1	1	X	1	X	0	X	0

Table 8.4.25

Step 3 : K-map simplification

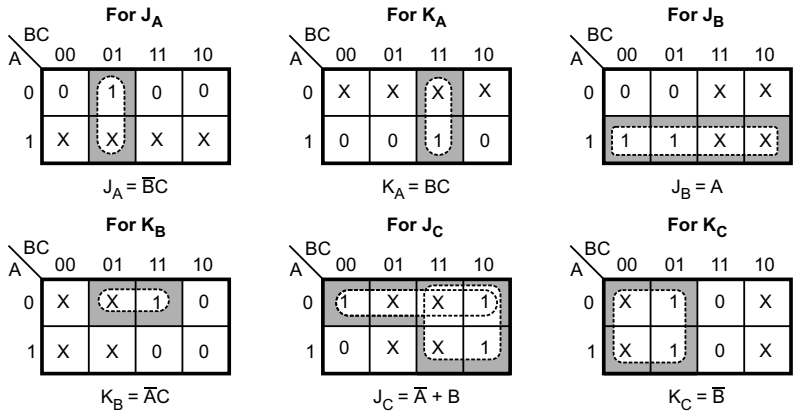


Fig. 8.4.31

Step 4 : Logic diagram

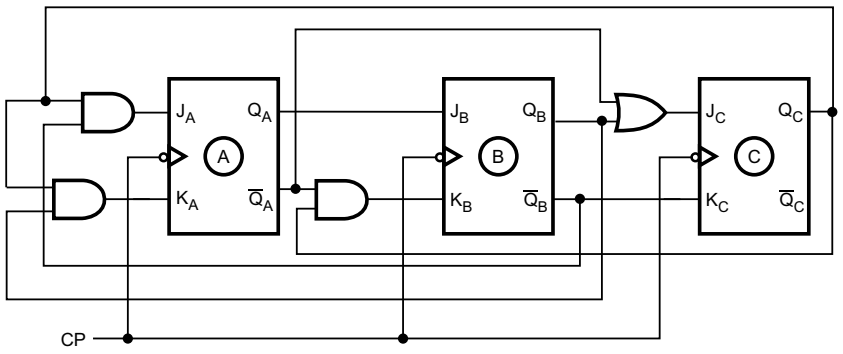


Fig. 8.4.32

Examples with Solutions

Example 8.4.12 *For a four bit even parity bit generator, inputs come serially. The four bits of the input sequence are to be examined by the circuit and circuit produces a parity bit which is to be added in the original sequence. The circuit should get ready for receiving another four bits after producing a parity bit for the last sequence. Draw the state diagram and write down the state transition table.* **May-08, Marks 10**

Solution :

Step 1 : Draw the state diagram.

The state diagram for given problem is as shown in the Fig. 8.4.33.

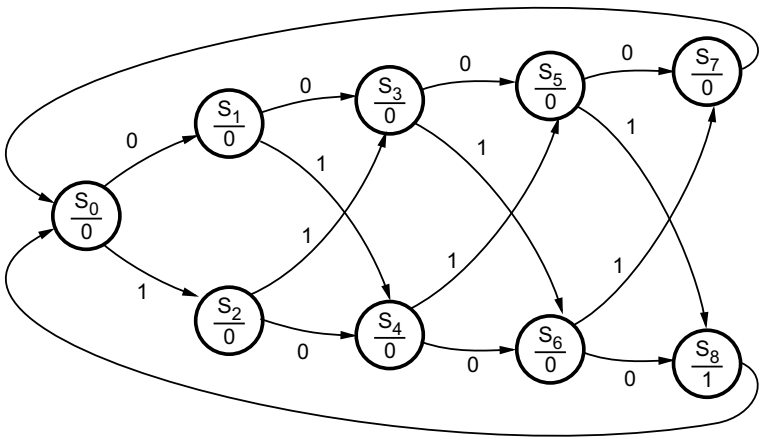


Fig. 8.4.33 State diagram

Step 2 : Determine the stable table.

Present state	Next state		Output
	x = 0	x = 1	z
S_0	S_1	S_2	0
S_1	S_3	S_4	0
S_2	S_4	S_3	0
S_3	S_5	S_6	0
S_4	S_6	S_5	0
S_5	S_7	S_8	0

S_6	S_8	S_7	0
S_7	S_0	S_0	0
S_8	S_0	S_0	1

Table 8.4.26 State table

Step 3 : Assign states, $S_0 = 0000$, $S_1 = 0001$, $S_2 = 0010$, $S_3 = 0011$, $S_4 = 0100$, $S_5 = 0101$, $S_6 = 0110$, $S_7 = 0111$, $S_8 = 1000$.

Step 4 : Determine the state transition table.

Present state	Next state		Output
	x = 0	x = 1	z
0000	0001	0010	0
0001	0011	0100	0
0010	0100	0011	0
0011	0101	0110	0
0100	0110	0101	0
0101	0111	1000	0
0110	1000	0111	0
0111	0000	0000	0
1000	0000	0000	1

Table 8.4.27 State transition table

Example 8.4.13 *Design a sequential circuit with 4 FF ABCD. The next states of B, C, D are equal to the present states of A, B, C respectively. The next state of A is equal to the EX-OR of the present states of C and D.* **May-07, 09, Marks 16**

Solution :

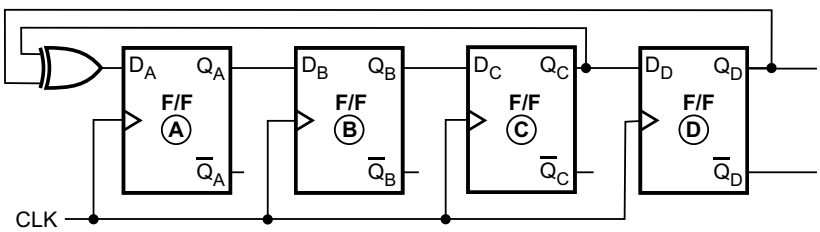


Fig. 8.4.34

Example 8.4.14 *Design a synchronous sequential circuit using JK for the given state diagram.*

May-10, Marks 16

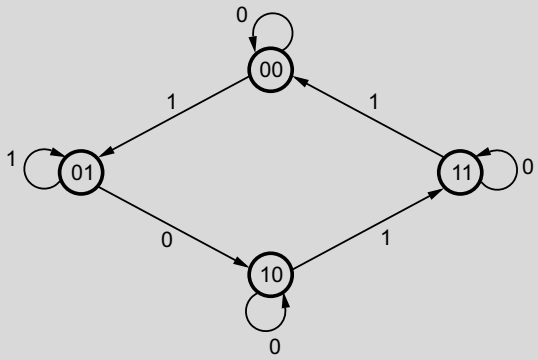


Fig. 8.4.35

Solution :

Step 1 : Since N = 4. Number of flip-flops needed = 2

Step 2 : Flip-flops to be used : JK

Step 3 : Determine the excitation table.

Present state		Input	Next state		Flip-flop inputs			
A	B	X	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Step 4 : K-map simplification

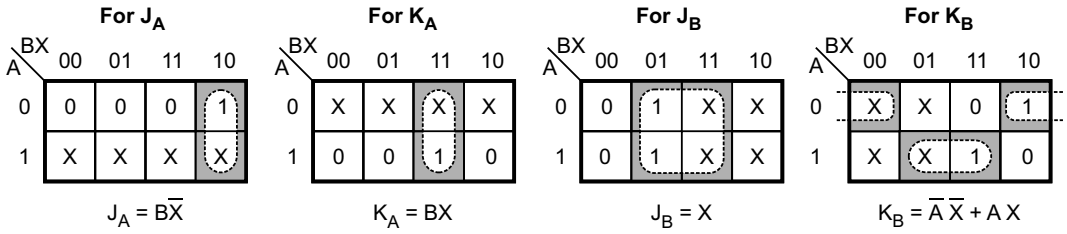


Fig. 8.4.36

Step 5 : Logic diagram :

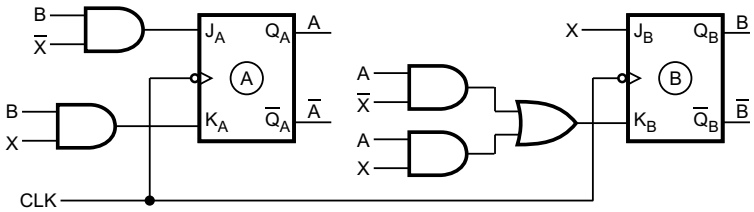


Fig. 8.4.37

Example 8.4.15 *With the help of two shift registers design a 4-bit Serial adder.*

Solution : Two shift registers are available to store the binary numbers to be added serially. The serial outputs from the registers are x and y. S gives the sum and carry after addition is stored in the flip-flop. Therefore Q output of flip-flop gives carry. Excitation table for a serial adder.

Present state	Inputs		Next state	Output	Flip-flop inputs	
Q	x	y	Q	S	J	K
0	0	0	0	0	0	X
0	0	1	0	1	0	X
0	1	0	0	1	0	X
0	1	1	1	0	1	X
1	0	0	0	1	X	1
1	0	1	1	0	X	0
1	1	1	1	0	X	0
1	1	1	1	1	X	0

Table 8.4.28

The circuit diagram is shown as

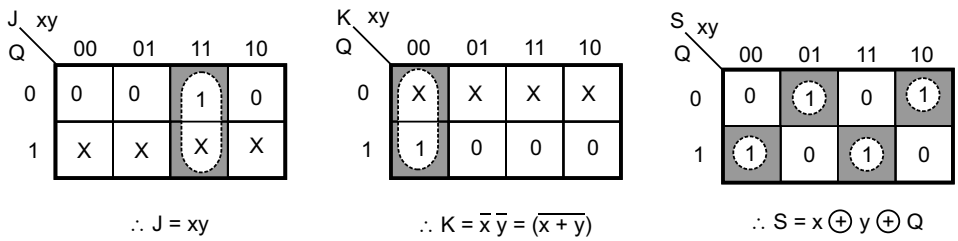


Fig. 8.4.38 (a)

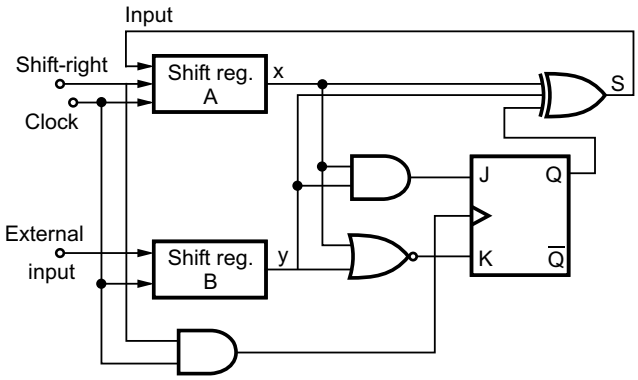


Fig. 8.4.38 (b)

Example 8.4.16 *Design the sequential circuit whose state table is given as*

Dec.-07, Marks 16

Present state		Input	Next state		Output
A_1	A_2	X	A_1	A_2	Y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

Solution : Let us design sequential circuit using D flip-flop.

Step 1 : K-map simplification

For A_1^+

A_1 \ A_2X	00	01	11	10
0	0	0	0	0
1	1	0	0	1

$A_1^+ = A_1 \overline{X}$

For A_2^+

A_1 \ A_2X	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$A_2^+ = \overline{A}_2 X + A_2 \overline{X}$

For Y

A_1 \ A_2X	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$Y = A_2 X$

Fig. 8.4.39 (a)

Step 2 : Logic diagram

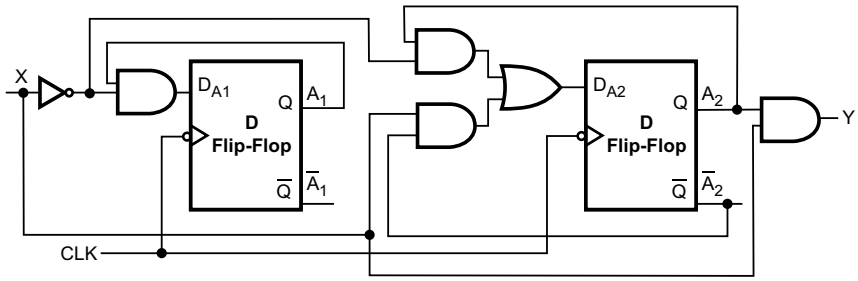


Fig. 8.4.39 (b)

Example 8.4.17 *Design a sequential circuit with JK flip flop to satisfy the following state equations*

$A(t + 1) = A'B'CD + A'B'C + ACD + AC'D'$

$B(t + 1) = A'C + CD' + A'BC'$

$C(t + 1) = B$

$D(t + 1) = D'$

Dec.-08, Marks 10

Solution :

Step 1 : Derive the state table from given state equations

Present state				Next state				Flip-flop inputs							
A	B	C	D	A+	B+	C+	D+	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	0	0	0	1	0	×	0	×	0	×	1	×
0	0	0	1	0	0	0	0	0	×	0	×	0	×	×	1
0	0	1	0	1	1	0	1	1	×	1	×	×	1	1	×
0	0	1	1	1	1	0	0	1	×	1	×	×	1	×	1
0	1	0	0	0	1	1	1	0	×	×	0	1	×	1	×
0	1	0	1	0	1	1	0	0	×	×	0	1	×	×	1
0	1	1	0	0	1	1	1	0	×	×	0	×	0	1	×
0	1	1	1	0	1	1	0	0	×	×	0	×	0	×	1
1	0	0	0	1	0	0	1	×	0	0	×	0	×	1	×
1	0	0	1	0	0	0	0	×	1	0	×	0	×	×	1

1	0	1	0	0	1	0	1	×	1	1	×	×	1	1	×
1	0	1	1	1	0	0	0	×	0	0	×	×	1	×	1
1	1	0	0	1	0	1	1	×	0	×	1	1	×	1	×
1	1	0	1	0	0	1	0	×	1	×	1	1	×	×	1
1	1	1	0	0	1	1	1	×	1	×	0	×	0	1	×
1	1	1	1	1	0	1	0	×	0	×	1	×	0	×	1

Note :

$A(t+1) = \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C + ACD + A\overline{C}\overline{D} = \overline{A}\,\overline{B}C + ACD + A\overline{C}\overline{D}$

$B(t+1) = \overline{A}C + C\overline{D} + \overline{A}B\overline{C} = \overline{A}C + \overline{A}B + C\overline{D}$

Table 8.4.29 State table

Step 1 : K-map simplification

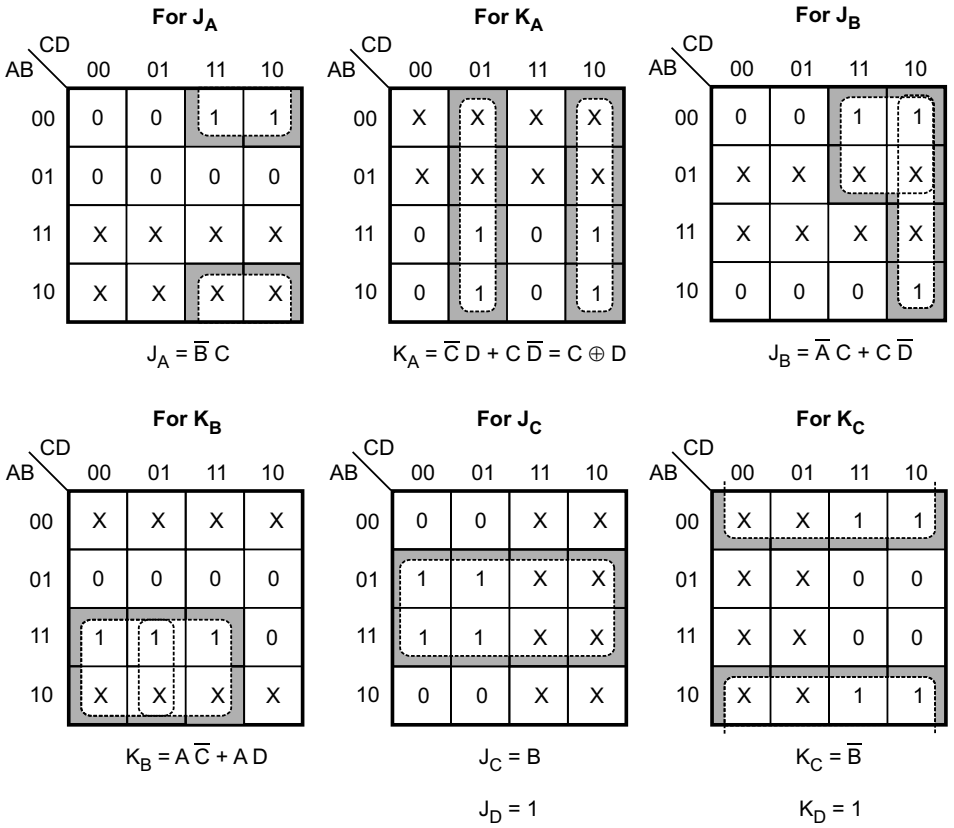


Fig. 8.4.40 (a)

Step 3 : Logic diagram

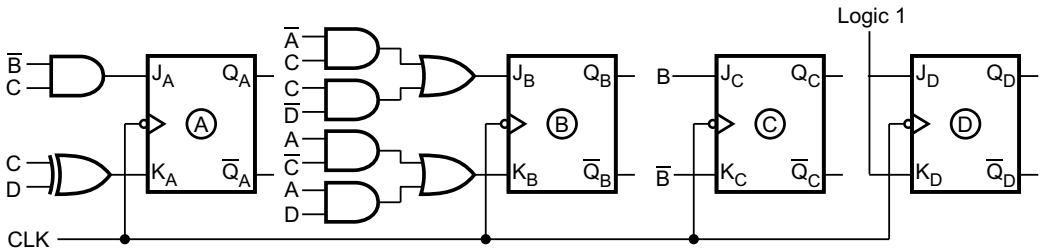


Fig. 8.4.40 (b)

Example 8.4.18 *For the state diagram shown in Fig. 8.4.41, design a synchronous sequential circuit using JK flip-flop.* **Dec.-10, Marks 16**

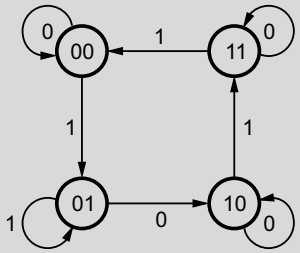


Fig. 8.4.41

Solution :

Step 1 : Excitation table

Input	Present state		Next state		Flip-Flop inputs			
X	A	B	A^+	B^+	J_A	K_A	J_B	K_B
0	0	0	0	0	0	X	0	X
0	0	1	1	0	1	X	X	1
0	1	0	1	0	X	0	0	X
0	1	1	1	1	X	0	X	0
1	0	0	0	1	0	X	1	X
1	0	1	0	1	0	X	X	0
1	1	0	1	1	X	0	1	X
1	1	1	0	0	X	1	X	1

Step 2 : K-map simplification

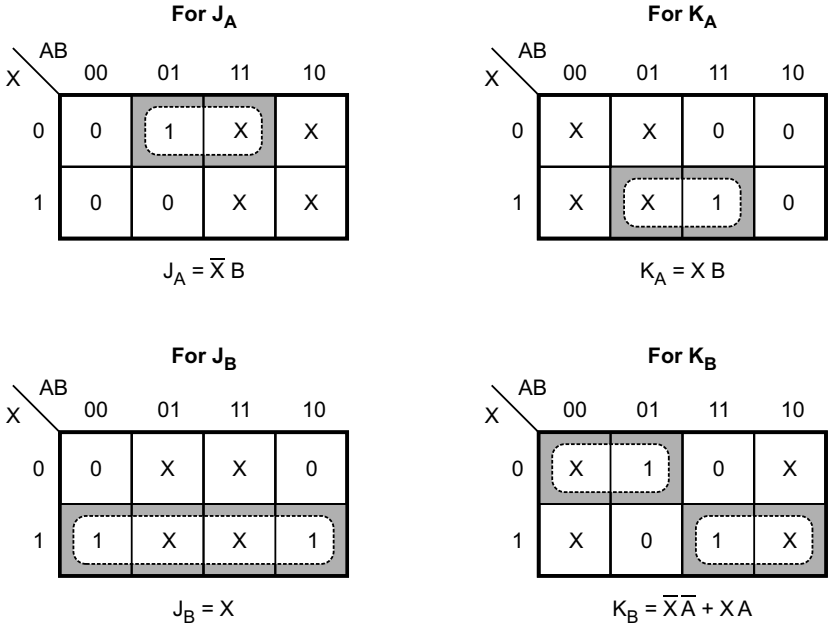


Fig. 8.4.41 (a)

Step 3 : Logic diagram

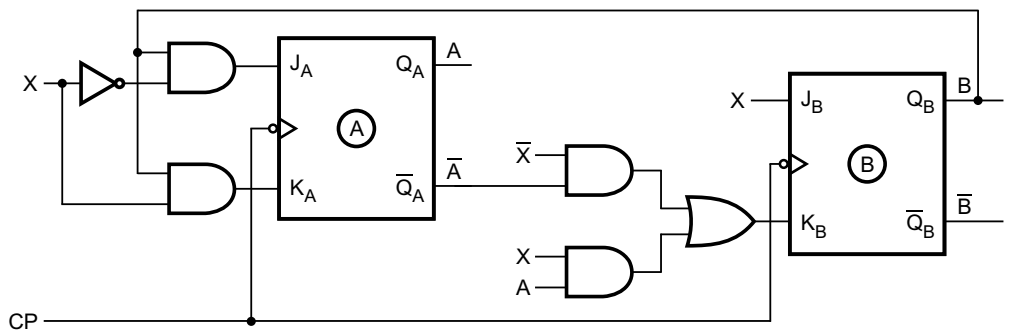


Fig. 8.4.41 (b)

Example 8.4.19 *Design a clocked sequential machine using T flip-flops for the following state diagram. Use state reduction if possible. Also use straight binary state assignment.*

May-11, Marks 8

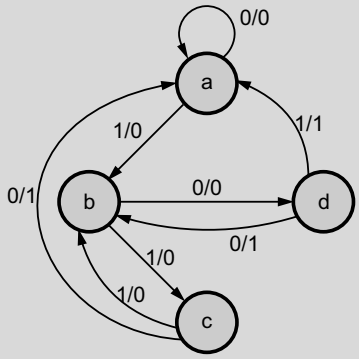


Fig. 8.4.42

Solution :

Step 1 : Derive state table.

State table for the given state diagram is a shown in Table 8.4.30 (a).

Present state	Next state		Output (Z)	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	d	c	0	0
c	a	b	1	0
d	b	a	1	1

Table 8.4.30 (a)

Step 2 : Derive excitation table.

Eventhough states a and c are having same next states for input X = 0 and X = 1, as the outputs are not same state reduction is not possible.

Using straight binary assignments as a = 00, b = 01, c = 10 and d = 11, the transition table (Excitations table) is as shown in Table 8.4.30 (b).

Input	Present state		Next state		Flip-flop input		Output
X	Q_A	Q_B	Q_{A+1}	Q_{B+1}	T_A	T_B	Z
0	0	0	0	0	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	1	0	1

0	1	1	0	1	1	0	1
1	0	0	0	1	0	1	0
1	0	1	1	0	1	1	0
1	1	0	0	1	1	1	0
1	1	1	0	0	1	1	1

Table 8.4.30 (b)

The flip-flop inputs and the circuit outputs are

$$T_A = f(Q_A, Q_B, X)$$

$$T_B = f(Q_A, Q_B, X)$$

$$Z = f(Q_A, Q_B, X)$$

Step 3 : K-map simplification

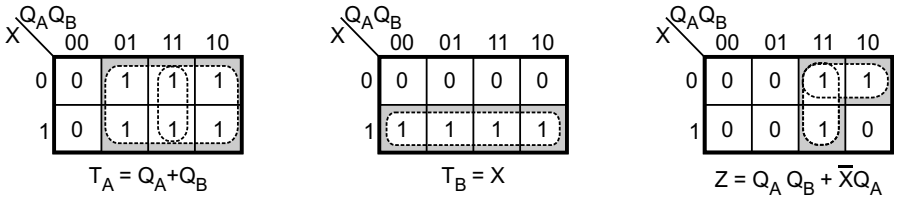


Fig. 8.4.42 (a)

Step 4 : Logic diagram

Therefore, the logic diagram is as shown in Fig. 8.4.42 (b).

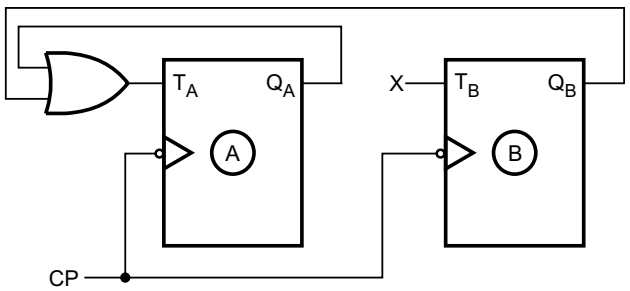


Fig. 8.4.42 (b)

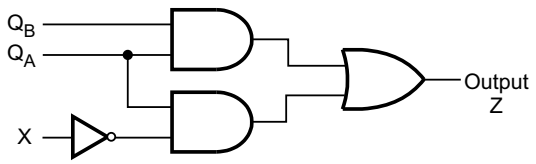


Fig. 8.4.42 (c)

Example 8.4.20 *Design a sequential circuit with two D flip-flops A and B and an input x. When x = 0 the state of the circuit remains the same. When x = 1 the circuit goes through the state transitions from 00 to 01 to 11 to 10 back to 00 and repeats.* **Dec.-11, Marks 10**

Solution :

Present state		Next state			
		x = 0		x = 1	
A	B	A^+	B^+	A^+	B^+
0	0	0	0	0	1
0	1	0	1	1	1
1	0	1	0	0	0
1	1	1	1	1	0

K-map simplification

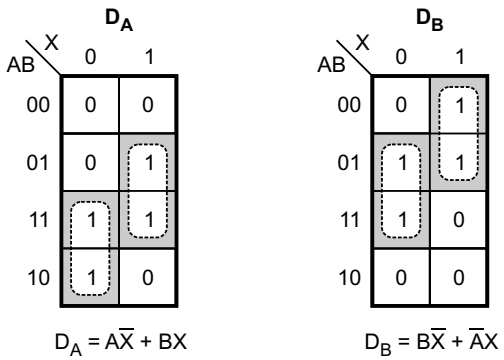


Fig. 8.4.43

Logic diagram

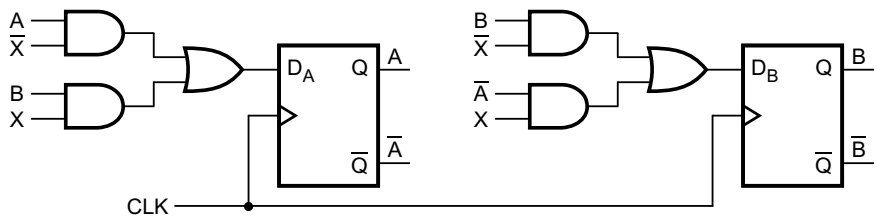


Fig. 8.4.43 (a)

Review Questions

1. *Give recommended steps for the design of a clocked synchronous sequential networks.*
2. *Why is state reduction necessary ?* **Dec.-06, May-09, Marks 2**
3. *Explain the state assignment rules.*
4. *What is lockout condition ?*
5. *What is a self starting counter ?* **May-10, Marks 2**
6. *Explain the procedure of state minimisation using merger graph and merger table.*
7. *Write note on state minimisation.* **Dec.-03, Marks 8**
8. *Define state assignment.* **Dec.-05, Marks 2**
9. *Explain the steps involved in the reduction of state table.* **Dec.-10, Marks 10**

8.5 Sequence Generator

8.5.1 Sequence Generator using Counters

A sequential circuit which generates a prescribed sequence of bits, in synchronism with a clock, is referred to as a sequence generator. Fig. 8.5.1 shows the basic structure of a sequence generator using counters.

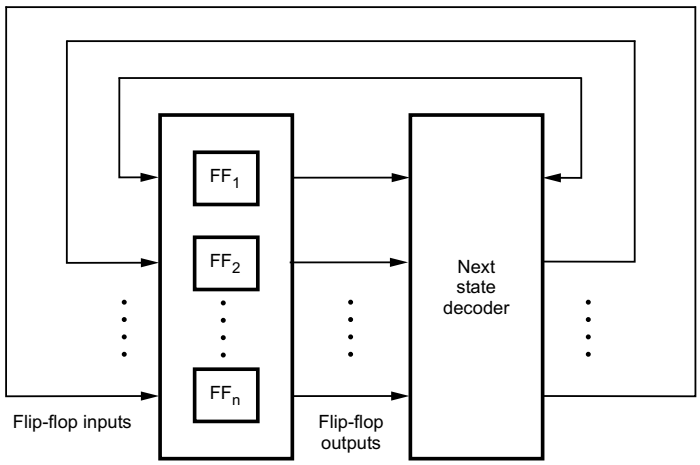


Fig. 8.5.1 Basic structure of a sequence generator

For the design of sequence generator, we must determine the required number of flip-flops and the logic circuit for next state decoder.

Number of flip-flops required

Number of flip-flops required to generate particular sequence can be determined as follows :

- Find the number of 1s in the sequence.

- Find the number of 0s in the sequence.
- Take the maximum out of two.
- If N is the required number of flip-flops, choose minimum value of n to satisfy equation given below.
  max (0s, 1s) $\leq 2^n - 1$

Examples with Solutions

Example 8.5.1 *Find the number of flip-flops required to generate the sequence 1101011.*

Solution : The given sequence number of 0s are 2 and number of 1s are 5. Therefore equation becomes,

$\therefore$ max (2, 5) $\leq 2^n - 1$

$\therefore$ $5 \leq 2^n - 1$

$\therefore$ $n = 4$

Once the number of flip-flops are decided, we have to assign unique states corresponding to each bit in the given sequence such that flip-flop representing least significant bit generates the given sequence. (Usually, the output of the flip-flop representing least significant bit is used to generate the given sequence).

Example 8.5.2 *Find the state assignments for sequence 1101011.*

Solution : We have already seen that this sequence requires four flip-flops. Assuming the output of D flip-flop as a desired sequence state assignments are shown in Table 8.5.1.

A	B	C	D	States
0	0	0	1	1
0	0	1	1	3
0	0	0	0	0
0	1	0	1	5
0	0	1	0	2
0	1	1	1	7
1	0	0	1	9

Table 8.5.1

Example 8.5.3 *Referring Table 8.5.1, draw state diagram and implement the sequence generator.*

Solution :

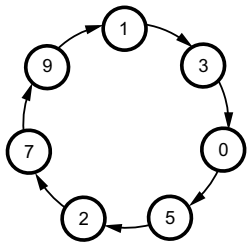


Fig. 8.5.2 State diagram

Present state				Next state				Flip-flop inputs							
Q_A	Q_B	Q_C	Q_D	Q_{A+1}	Q_{B+1}	Q_{C+1}	Q_{D+1}	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	0	1	0	1	0	X	1	X	0	X	1	X
0	0	0	1	0	0	1	1	0	X	0	X	1	X	X	0
0	0	1	0	0	1	1	1	0	X	1	X	X	0	1	X
0	0	1	1	0	0	0	0	0	X	0	X	X	1	X	1
0	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
0	1	0	1	0	0	1	0	0	X	X	1	1	X	X	1
0	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
0	1	1	1	1	0	0	1	1	X	X	1	X	1	X	0
1	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	0	1	0	0	0	1	X	1	0	X	0	X	X	0
1	0	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X

Table 8.5.2

K-map simplification

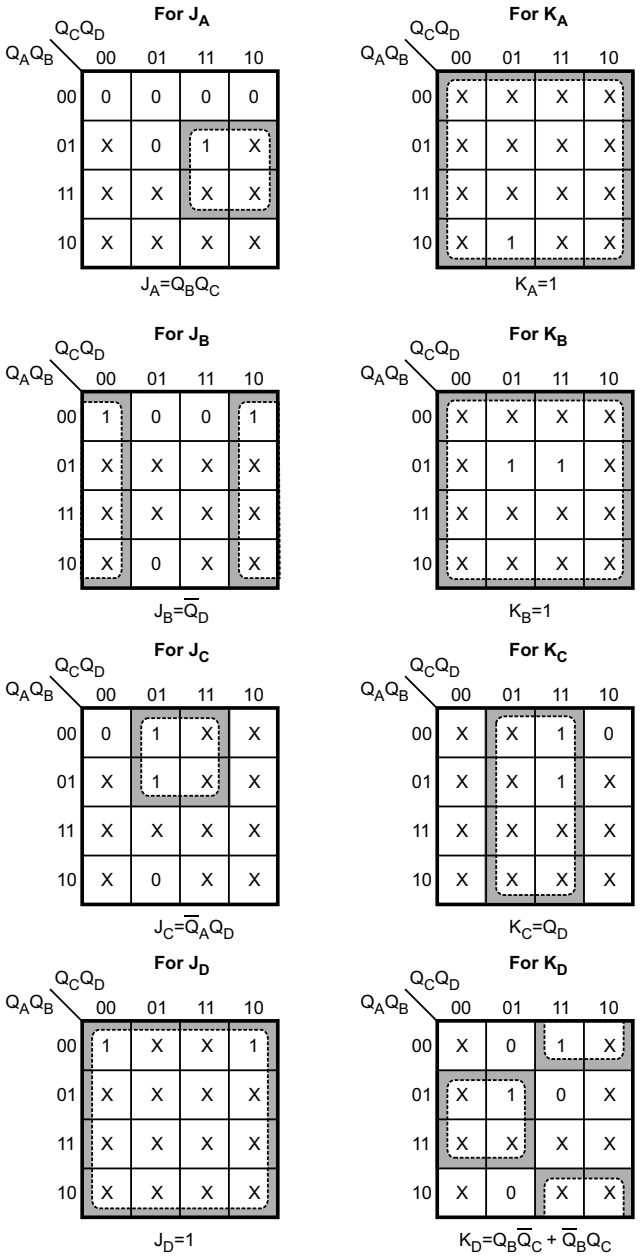


Fig. 8.5.3

Logic diagram

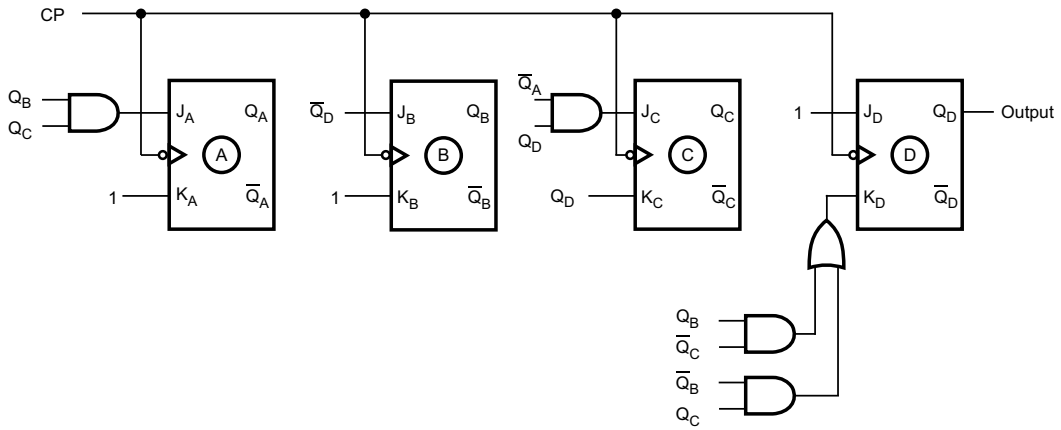


Fig. 8.5.4

Example 8.5.4 *Design synchronous counter using D flip-flop for the sequence as shown in the Fig. 8.5.5.*

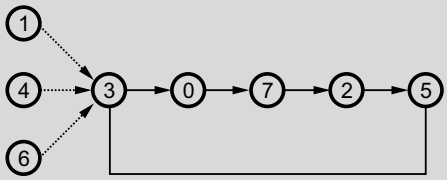


Fig. 8.5.5

Solution : Flip-flop excitation table is as shown below.

Present state			Next state		
A	**B**	**C**	A_{+1}	B_{+1}	C_{+1}
0	0	0	1	1	1
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	0	1	0

Table 8.5.3

K-map simplification

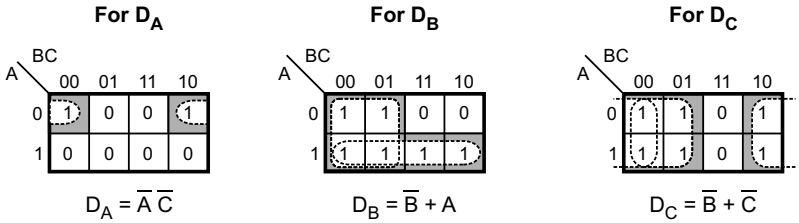


Fig. 8.5.6

Logic diagram

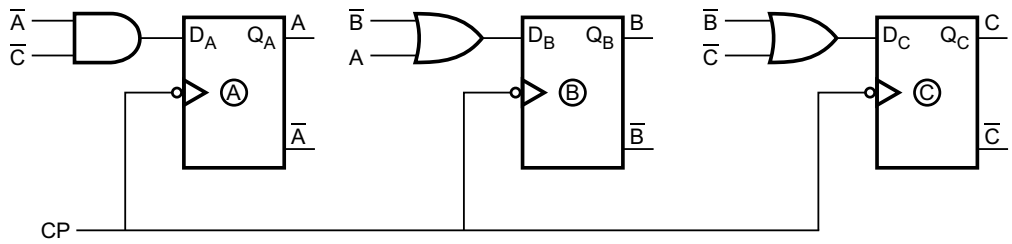


Fig. 8.5.7

Example 8.5.5 *Using J-K flip-flops, design a synchronous counter that has the following sequence :*

..... 0 → 2 → 5 → 6 → 0 .....

undesired states 1, 3, 4, 7 must always go to 0 on the next clock pulse.

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
A	B	C	A_{+1}	B_{+1}	C_{+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	0	0	0	0	X	0	X	X	1
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	0	0	0	X	1	X	1	X	1

Table 8.5.4

K-map simplification

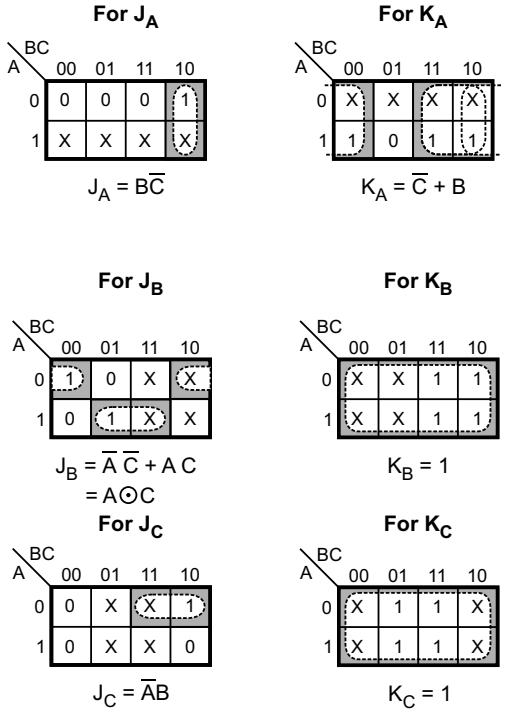


Fig. 8.5.8

Logic diagram

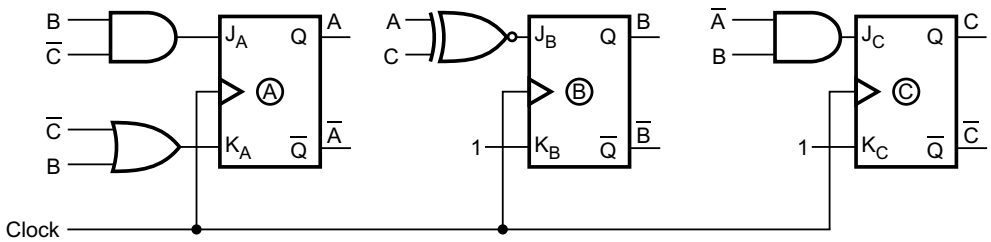


Fig. 8.5.9

Example 8.5.6 *Design and implement 4-bit binary counter (using D flip-flops) which counts all possible odd numbers only.*

Solution : The state diagram for given problem is as shown in the Fig. 8.5.10.

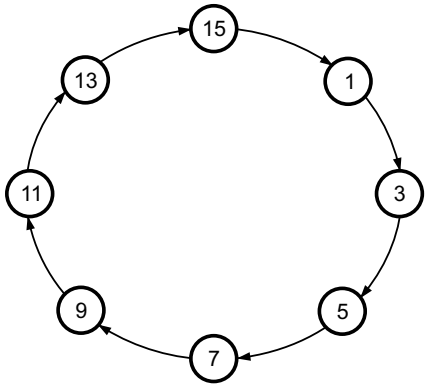


Fig. 8.5.10 State diagram

Excitation table

Present state				Next state			
Q_A	Q_B	Q_C	Q_D	Q_{A+}	Q_{B+}	Q_{C+}	Q_{D+}
0	0	0	0	×	×	×	×
0	0	0	1	0	0	1	1
0	0	1	0	×	×	×	×
0	0	1	1	0	1	0	1
0	1	0	0	×	×	×	×
0	1	0	1	0	1	1	1
0	1	1	0	×	×	×	×
0	1	1	1	1	0	0	1
1	0	0	0	×	×	×	×
1	0	0	1	1	0	1	1
1	0	1	0	×	×	×	×
1	0	1	1	1	1	0	1
1	1	0	0	×	×	×	×
1	1	0	1	1	1	1	1
1	1	1	0	×	×	×	×
1	1	1	1	0	0	0	1

K-map simplification

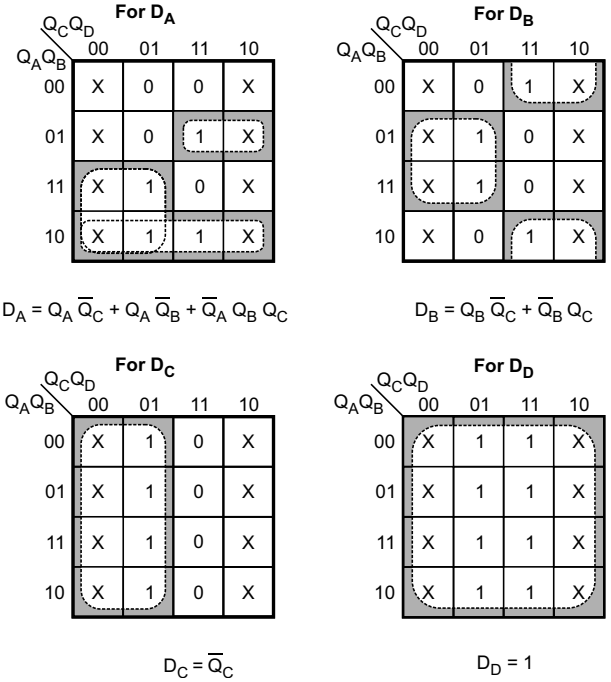

For D_A
Q_CQ_D 00 01 11 10
Q_AQ_B
00 X 0 0 X
01 X 0 1 X
11 X 1 0 X
10 X 1 1 X
$D_A = Q_A \overline{Q}_C + Q_A \overline{Q}_B + \overline{Q}_A Q_B Q_C$
For D_B
Q_CQ_D 00 01 11 10
Q_AQ_B
00 X 0 1 X
01 X 1 0 X
11 X 1 0 X
10 X 0 1 X
$D_B = Q_B \overline{Q}_C + \overline{Q}_B Q_C$
For D_C
Q_CQ_D 00 01 11 10
Q_AQ_B
00 X 1 0 X
01 X 1 0 X
11 X 1 0 X
10 X 1 0 X
$D_C = \overline{Q}_C$
For D_D
Q_CQ_D 00 01 11 10
Q_AQ_B
00 X 1 1 X
01 X 1 1 X
11 X 1 1 X
10 X 1 1 X
$D_D = 1$

Fig. 8.5.11

Logic diagram

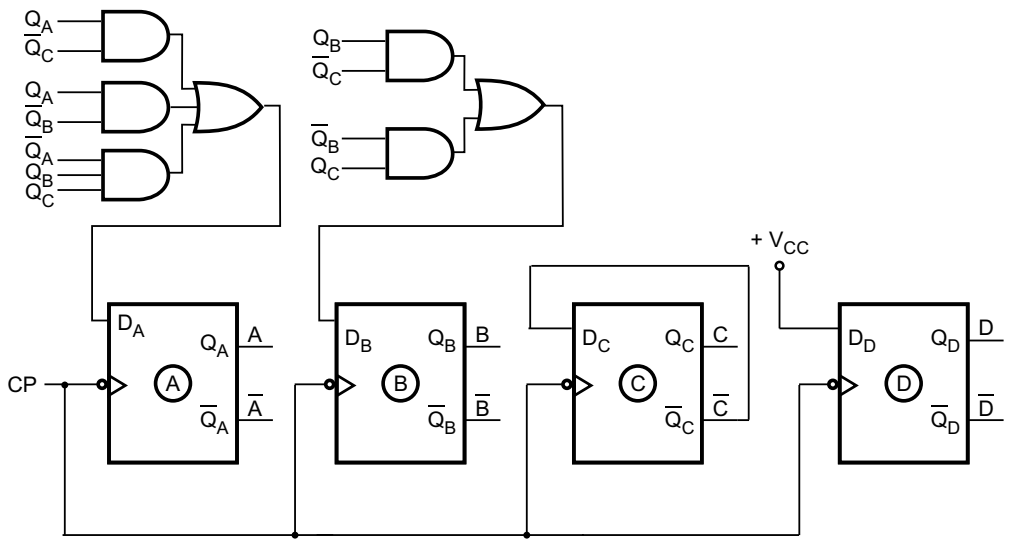


Fig. 8.5.12

Example 8.5.7 *Design a pulse-train generator to generate a pulse train 110011..... using 'D' flip-flop.*

Solution : The minimum number of flip-flops can be given as $N \leq 2^n - 1$.

Here N = 6, therefore n = 3. Table 8.5.5 shows state encoding for given sequence using 3 flip-flops.

CP	FF O/Ps			States
	Q_A	Q_B	Q_C	
1	1	1	1	7
2	1	1	0	6
3	0	0	1	1
4	0	0	0	0
5	1	0	0	4
6	1	0	1	5

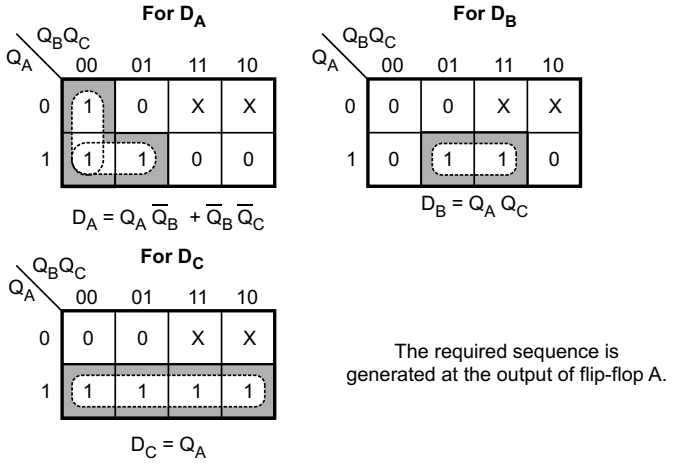
Table 8.5.5

Therefore, design a circuit to get above state. In case of D flip-flop, flip-flop inputs are same as next states.

Excitation table

Present state			Next state/Flip-flop inputs		
Q_A	Q_B	Q_C	Q_A	Q_B	Q_C
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	x	x	x
0	1	1	x	x	x
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	1	1

K-map simplification



Logic diagram

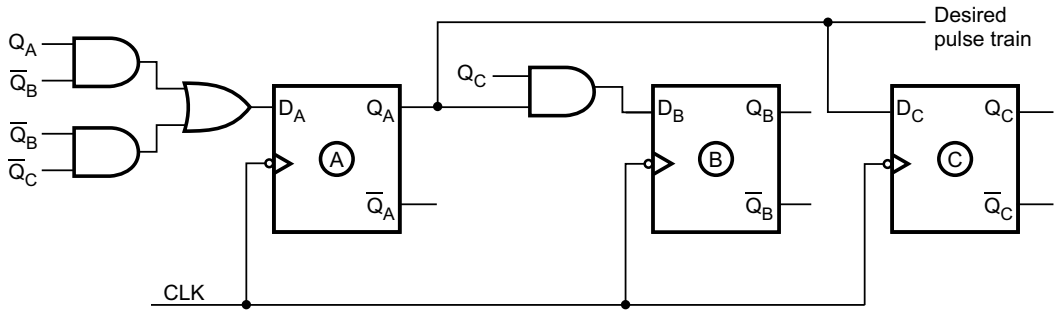


Fig. 8.5.13

Example 8.5.8 *Design a non-sequential counter using J-K flip-flop; as per following state diagram.*

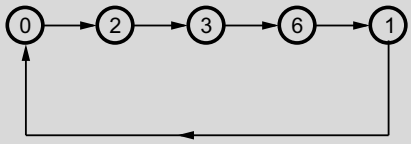


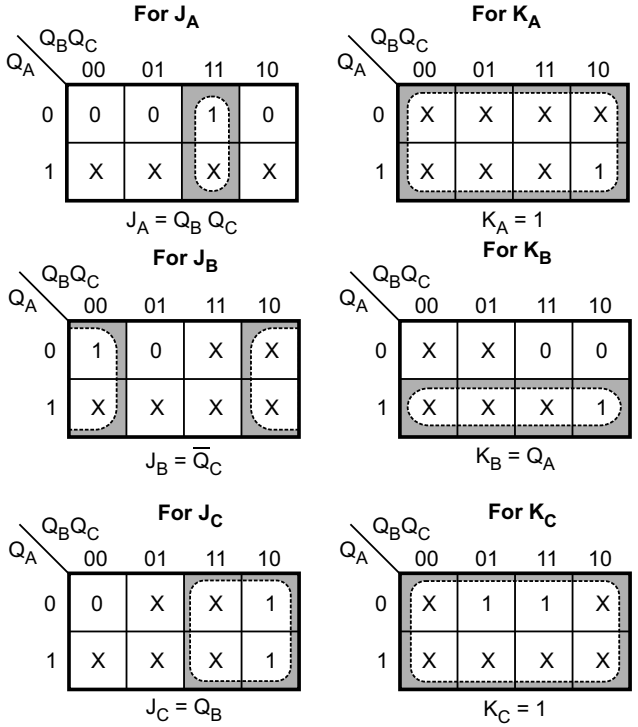
Fig. 8.5.14

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
Q_A	Q_B	Q_C	Q'_A	Q'_B	Q'_C	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	0	0	x	1	x	0	x
0	0	1	0	0	0	0	x	0	x	x	1

0	1	0	0	1	1	0	x	x	0	1	x
0	1	1	1	1	0	1	x	x	0	x	1
1	0	0	x	x	x	x	x	x	x	x	x
1	0	1	x	x	x	x	x	x	x	x	x
1	1	0	0	0	1	x	1	x	1	1	x
1	1	1	x	x	x	x	x	x	x	x	x

K-map simplification



Logic diagram

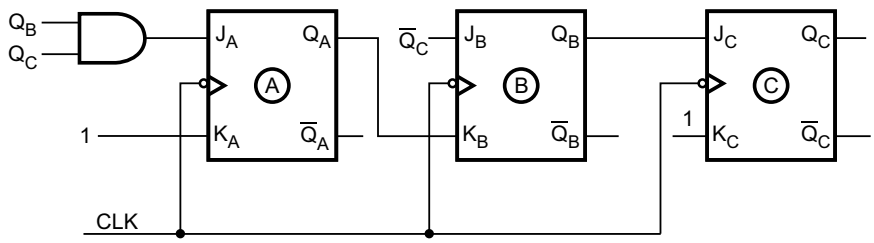


Fig. 8.5.14 (a)

Example 8.5.9 *Design the circuit to generate the sequence :*
$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$.

Solution : Excitation table

Present state			Next state			Flip-flop inputs					
Q_A	Q_B	Q_C	Q_{A+1}	Q_{B+1}	Q_{C+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	1	0	0	X	1	X	0	X
0	0	1	X	X	X	X	X	X	X	X	X
0	1	0	1	0	1	1	X	X	1	1	X
0	1	1	0	0	0	0	X	X	1	X	1
1	0	0	1	1	1	X	0	1	X	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	0	1	1	X	1	X	0	X	0

K-map simplification

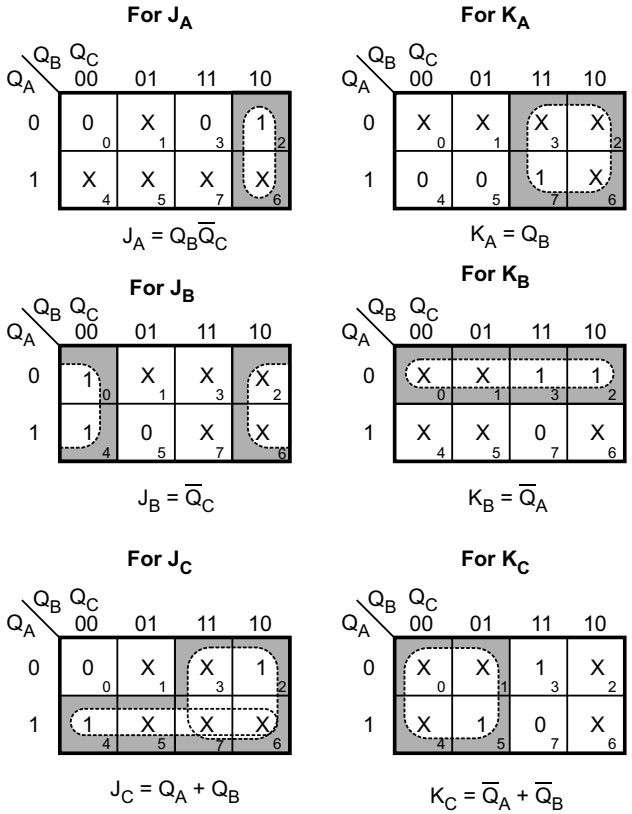


Fig. 8.5.15 (a)

Logic diagram

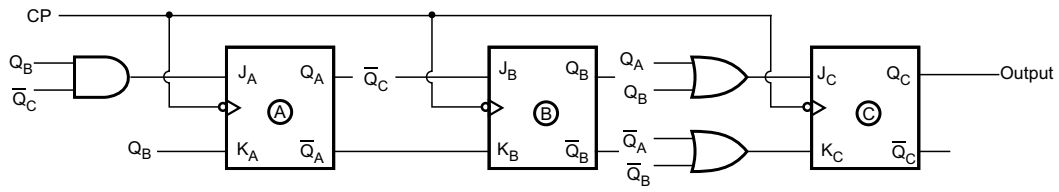


Fig. 8.5.15 (b)

Example 8.5.10 *Design sequence generator to generate sequence 1-9-2-7-3-6 using JK flip-flop.*
$0 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 3$.

Solution : Excitation table

Present state				Next state				Flip-flop inputs							
Q_A	Q_B	Q_C	Q_D	Q_{A+1}	Q_{B+1}	Q_{C+1}	Q_{D+1}	J_A	K_A	J_B	K_B	J_C	K_C	J_D	K_D
0	0	0	0	×	×	×	×	×	×	×	×	×	×	×	×
0	0	0	1	1	0	0	1	1	×	0	×	0	×	×	0
0	0	1	0	0	1	1	1	0	×	1	×	×	0	1	x
0	0	1	1	0	1	1	0	0	×	1	×	×	0	×	1
0	1	0	0	×	×	×	×	×	×	×	×	×	×	×	×
0	1	0	1	×	×	×	×	×	×	×	×	×	×	×	×
0	1	1	0	0	0	0	1	0	×	×	1	×	1	1	×
0	1	1	1	0	0	1	1	0	×	×	1	×	0	×	0
1	0	0	0	×	×	×	×	×	×	×	×	×	×	×	×
1	0	0	1	0	0	1	0	×	1	0	×	1	×	×	1
1	0	1	0	×	×	×	×	×	×	×	×	×	×	×	×
1	0	1	1	×	×	×	×	×	×	×	×	×	×	×	×
1	1	0	0	×	×	×	×	×	×	×	×	×	×	×	×
1	1	0	1	×	×	×	×	×	×	×	×	×	×	×	×
1	1	1	0	×	×	×	×	×	×	×	×	×	×	×	×
1	1	1	1	×	×	×	×	×	×	×	×	×	×	×	×

K-map simplification

For J_A

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	1	0	0
01	X	X	0	0
11	X	X	X	X
10	X	X	X	X

$J_A = \overline{Q}_C$

For K_A

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10	X	1	X	X

$K_A = 1$

For J_B

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	0	1	1
01	X	X	X	X
11	X	X	X	X
10	X	0	X	X

$J_B = Q_C$

For K_B

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	X	X	X
01	X	X	1	1
11	X	X	X	X
10	X	X	X	X

$K_B = 1$

For J_C

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	0	X	X
01	X	X	X	X
11	X	X	X	X
10	X	1	X	X

$J_C = Q_A$

For K_C

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	X	0	0
01	X	X	0	1
11	X	X	X	X
10	X	X	X	X

$K_C = Q_B\overline{Q}_D$

For J_D

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	X	X	1
01	X	X	X	1
11	X	X	X	X
10	X	X	X	X

$J_D = 1$

For K_D

Q_AQ_B \ Q_CQ_D	00	01	11	10
00	X	0	1	X
01	X	X	0	X
11	X	X	X	X
10	X	1	X	X

$K_D = Q_A + \overline{Q}_BQ_C$

Logic diagram

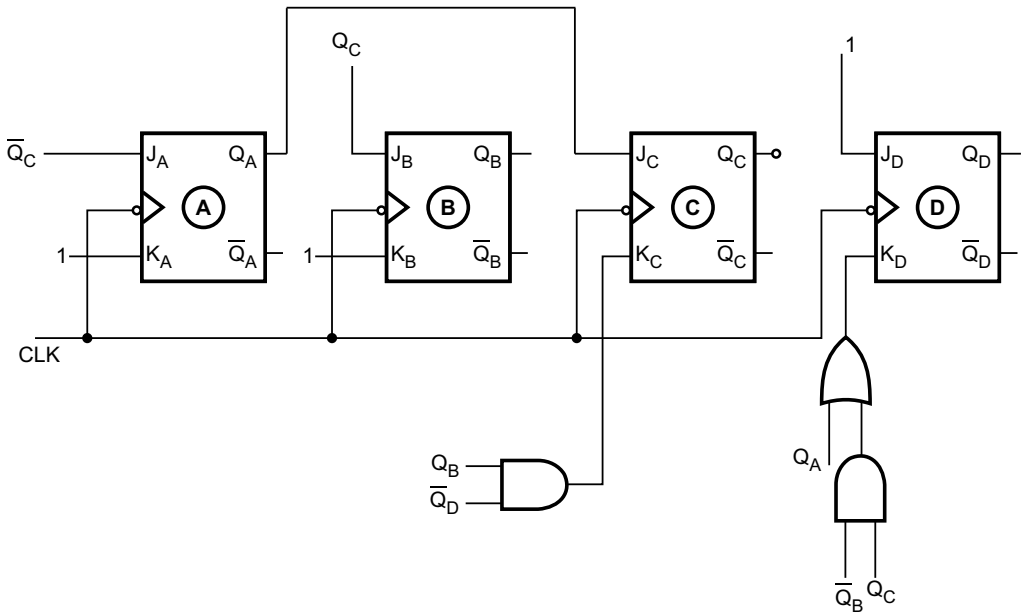


Fig. 8.5.16

8.5.2 Sequence Generator using Shift Register

The simplest way of designing sequence generator using shift register is to take shift register of n-bits where n is equal to the length of sequence. Then load the bit sequence in the shift register by parallel load operation and apply the clock signal. The Fig. 8.5.17 illustrates the operation of such a sequence generator. Here, the sequence to be generated is 11001.

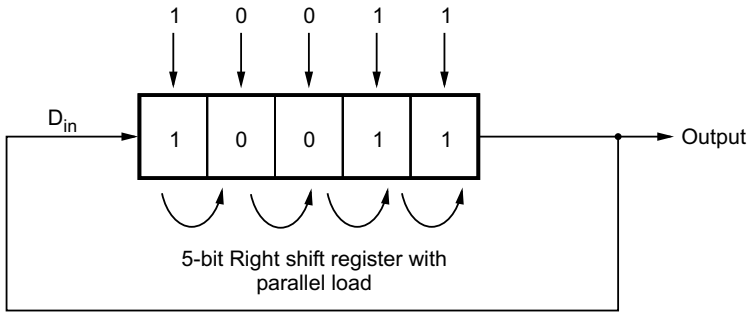


Fig. 8.5.17 Sequence generator

Another design approach is used for sequence generator to reduce the required number of flip-flop stages in the shift register. In this approach a shift register with a next state decoder and preset logic is used. The Fig. 8.5.18 shows the block diagram of

this approach. Here, the output of the next state decoder is a function of Q_A, Q_B, ... Q_n and it is used to determine the D_{in} input for the shift right register. Initially, start button is pressed to activate parallel load operation. This loads initial value in the shift register. Then at each clock pulse shift register contents are shifted right by 1 bit position. The next state decoder circuit decodes the output and generates D_{in} input for the shift register such that output Q_A generates the desired sequence.

After completion of one complete sequence, the register is again loaded with initial value to start the next train of sequence.

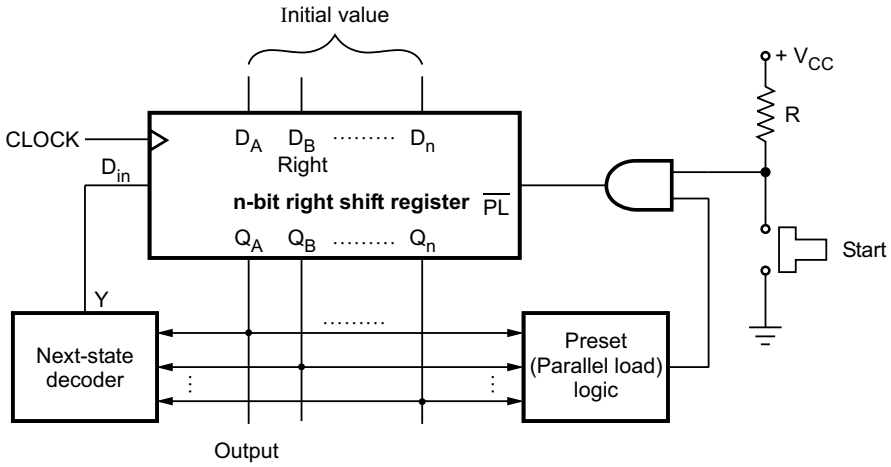


Fig. 8.5.18 Sequence generator using shift register

Examples with Solutions

Example 8.5.11 *Design a sequence generator to generate the sequence 1101011 by shift register method.*

Solution : In this approach, the minimum number of flip-flops n, required to generate a sequence of length N is given by

$$N \leq 2^n - 1$$

In this example, N = 7, therefore, the minimum value of n, which may generate this sequence is 3. However, it is not guaranteed to lead to a solution. Let us try with 3 flip-flops. The Table 8.5.6 shows sequence generation with three flip-flops.

	CP	Flip-flop outputs			D_{in}	States
		Q_A	Q_B	Q_C		
	1	1	0	0	1	4
	2	1	1	0	0	6
	3	0	1	1	1	3
	4	1	0	1	0	5
	5	0	1	0	1	2
State is repeated →	6	1	0	1	–	5
	–	–	–	–	–	–

Table 8.5.6

As shown in the Table 8.5.7, the state 6 is repeated. This means that n = 3 is not sufficient. Let us try with 4 flip-flops. The table shows sequence generation with four flip-flops.

	CP	Flip-flop outputs				D_{in}	$\overline{Preset}$	States
		Q_A	Q_B	Q_C	Q_D			
Initial value →	1	1	0	0	0	1	1	8
	2	1	1	0	0	0	1	12
	3	0	1	1	0	1	1	6
	4	1	0	1	1	0	1	11
	5	0	1	0	1	1	1	5
	6	1	0	1	0	1	1	10
	7	1	1	0	1	1	1	13
Preset flip-flop	1	1	1	1	0	X	0	14
	1	1	0	0	0	1	1	8

Table 8.5.7

As shown in the Table 8.5.7, states are not repeated. After completion of one complete sequence register is preset to value 1000 to start the next train of sequence. Fig. 8.5.19 (b) shows the logic diagram.

K-map simplification for D_{in}

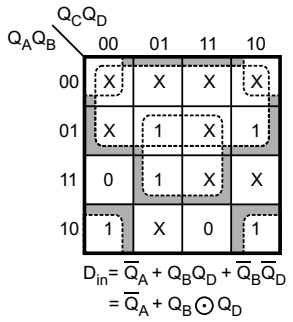


Fig. 8.5.19 (a)

Logic diagram

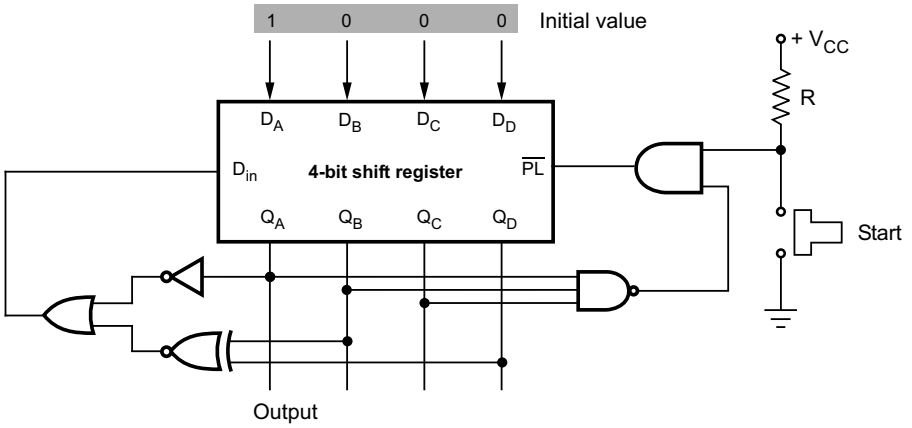


Fig. 8.5.19 (b)

Example 8.5.12 *Design and implement the following sequence generator using shift register .... 1010 .....*

Solution : Here the sequence is 10 and it is repeated. Thus the length of the sequence N = 2. Let the number of flip-flops required n be.

$$N \leq 2^n - 1$$

$$2 \leq 2^n - 1$$

$$\therefore \quad \mathbf{n = 2}$$

CP	Flip-flop outputs		D_{in}	States
	Q_A	Q_B		
1	1	0	0	2
2	0	1	1	1
1	1	0	0	2

K-map simplification for D_{in}

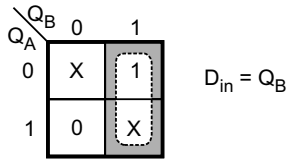


Fig. 8.5.20 (a)

Logic diagram

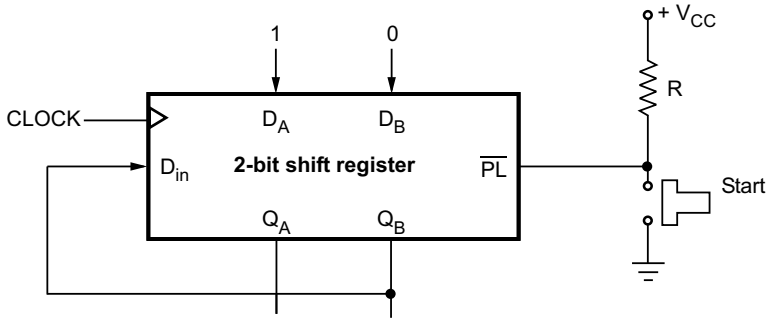


Fig. 8.5.20 (b)

In the above circuit preset/parallel load logic is only required to load the initial value. After completion of one sequence, the shift register have the same initial value and hence it is not required to reload again.

Example 8.5.13 *Design a pulse train generator using shift register for the following pulse train.*

.....1 1 1 0 1 0 .....

Solution :

The minimum number of flip-flops n can be given as

$$N \leq 2^n - 1$$

Here N = 6, therefore n = 3. The Table 8.5.8 shows sequence generation with three flip-flops.

CP	Flip-flop outputs			D_{in}	$\overline{Preset}$	States
	Q_A	Q_B	Q_C			
1	1	0	0	1	1	4
2	1	1	0	1	1	6
3	1	1	1	0	1	7

	4	0	1	1	1	1	3
	5	1	0	1	0	1	5
	6	0	1	0	0	1	2
Preset flip-flop	7	0	0	1	X	0	1
	1	1	0	0	1	1	4

Table 8.5.8

K-map simplification for D_{in}

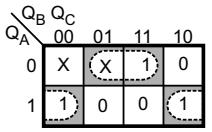


Fig. 8.5.21 (a)

$$D_{in} = \overline{Q}_A Q_C + Q_A \overline{Q}_C$$
$$= Q_A \oplus Q_C$$

Logic diagram

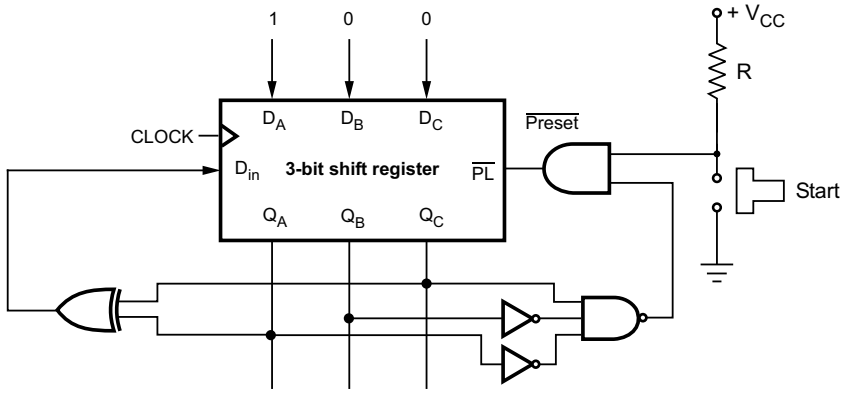


Fig. 8.5.21 (b)

Example 8.5.14 *Design a pulse train generator circuit using shift register for the following pulse train.*

..... 1 0 0 0 1 1 0.....

Solution : The minimum number of flip-flops n, required to generate sequence of length N is given by

$$N \leq 2^n - 1$$

Here, N = 7, therefore n = 3. The Table 8.5.9 (a) shows sequence generation with three flip-flops.

	CP	Flip-flop outputs			D_{in}	States
		Q_A	Q_B	Q_C		
	1	1	1	1	0	7
	2	0	1	1	0	3
	3	0	0	1	0	1
	4	0	0	0	1	0
	5	1	0	0	1	4
	6	1	1	0	0	6
State is repeated	7	0	1	1	–	3

Table 8.5.9 (a)

As seventh state is repeated, n = 3 is not sufficient. Let us try with 4 flip-flops. The Table 8.5.9 (b) shows sequence generation with four flip-flops.

CP	Flip-flop outputs				D_{in}	$\overline{\text{Preset}}$	States
	Q_A	Q_B	Q_C	Q_D			
1	1	1	1	1	0	1	15
2	0	1	1	1	0	1	7
3	0	0	1	1	0	1	3
4	0	0	0	1	1	1	1
5	1	0	0	1	1	1	9
6	1	1	0	0	0	1	12
7	0	1	1	0	1	1	6
Preset flip-flop 1	1	0	1	1	X	0	11
1	1	1	1	1	0	1	15

Table 8.5.9 (b)

K-map simplification for D_{in}

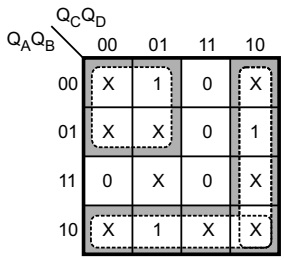


Fig. 8.5.22

$$\therefore \quad D_{in} = \overline{Q}_A \overline{Q}_C + Q_A \overline{Q}_B + Q_C \overline{Q}_D$$

Logic diagram

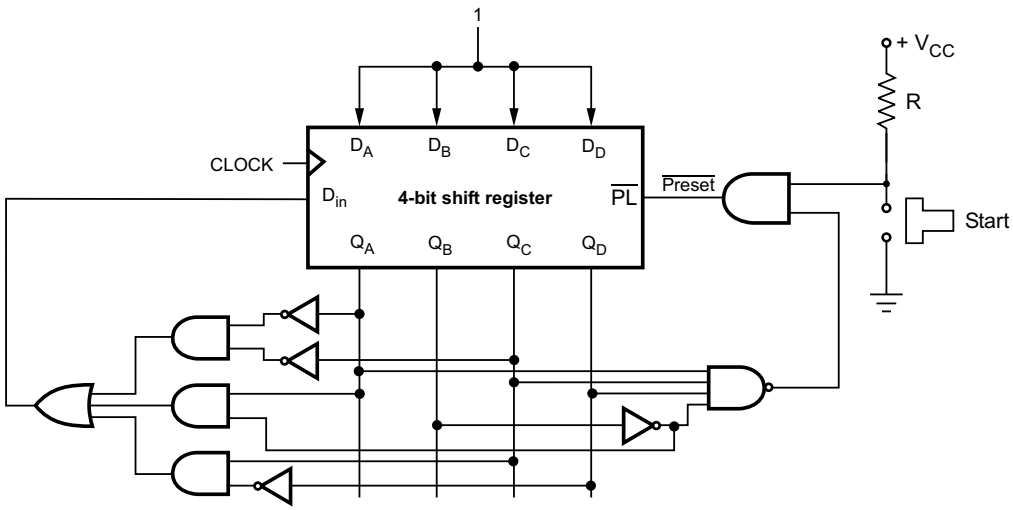


Fig. 8.5.23

Example 8.5.15 *Design sequence generator using shift register to generate sequence 1101.*

Solution : The minimum number of flip-flops n can be given as

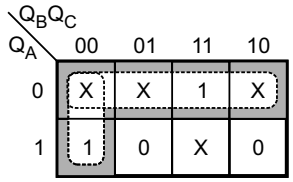
$$N \leq 2^n - 1$$

Here N = 4, therefore n = 3. The Table 8.5.10 shows sequence generation with three flip-flops.

	CP	Flip-flop inputs			D_{in}	$\overline{PL}$	States
		Q_A	Q_B	Q_C			
	1	1	0	0	1	1	4
	2	1	1	0	0	1	6
	3	0	1	1	1	1	3
	4	1	0	1	0	1	5
Preset flip-flop	5	0	1	0	×	0	2
	1	1	0	0	1	1	4

Table 8.5.10

K-map simplification for D_{in}



$D_{in} = \overline{Q}_B\overline{Q}_C + \overline{Q}_A$

Fig. 8.5.24 (a)

Logic diagram

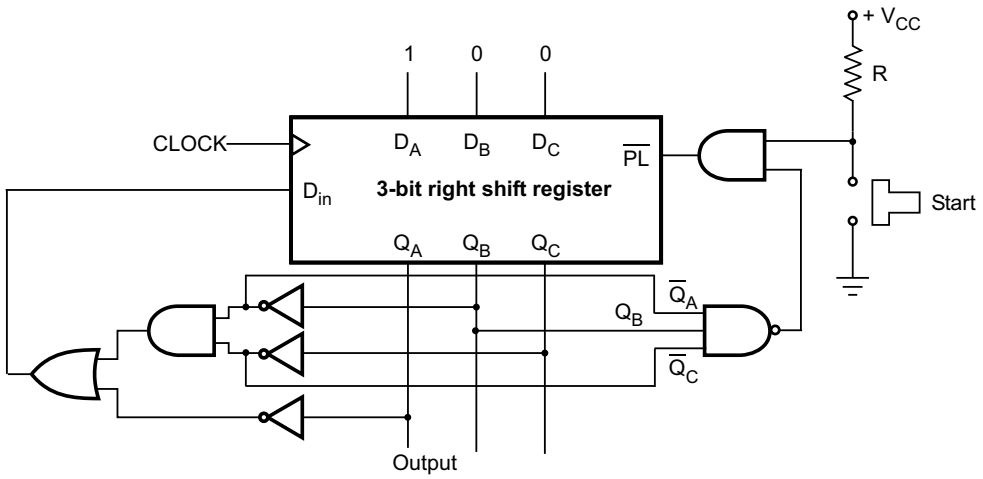
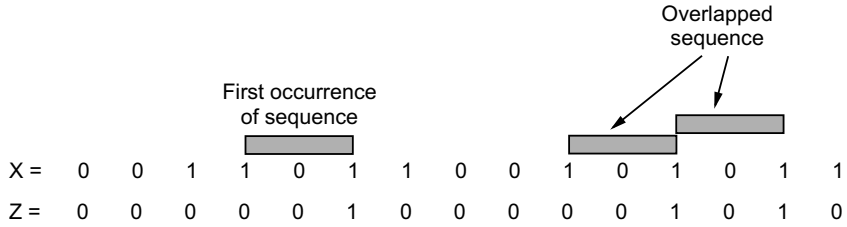


Fig. 8.5.24 (b)

8.6 Sequence Detector

Dec.-12

The specified input sequence can be detected using a sequential machine called **sequence detector**. In this circuit output goes high when a prescribed input sequence occurs. A typical input sequence and the corresponding output sequence for desired input sequence 101 are :



As shown above the detection of required input sequence can occur in a longer data string and the desired input sequence can overlap with another input sequence. It is assumed that input can change only between clock pulses. Once we know the sequence which is to be detected, we can draw the state diagram for it. Then from the state diagram we can design the circuit. It is possible to implement sequence detector using both types of sequential machines : Mealy machine and Moore machine. The following examples illustrate how to determine the state diagram from the given input sequence and then implement the sequence detector.

Illustrative Examples

Example 8.6.1 *Design a Mealy type sequence detector to detect a serial input sequence of 101.*

Solution : In a Mealy type design, the number of states in the state diagram are equal to the number of bits in the desired input sequence. In this example, we have three bits in the sequence so we have three states in the state diagram. Once the number of states are known we have to draw the direction lines with states of inputs and outputs. Let us start drawing direction lines, assuming state 'a' as an initial state.

1. State a

State a checks for 1. When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence.

When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0 since we have not yet detected all the bits in the sequence.

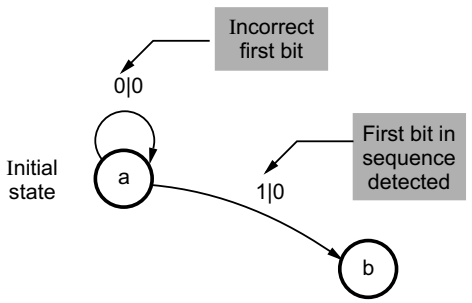


Fig. 8.6.1 (a)

2. State b

When input is 0, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 1, we have to remain in state b because 1 which we have detected may start the sequence. Output is still zero.

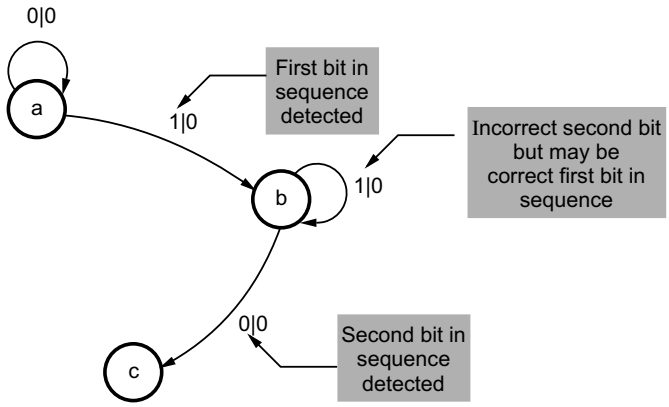


Fig. 8.6.1 (b)

3. State c

As explained for state a and state b, if desired bit is detected we have to go for next state otherwise we have to go to the previous state from where we can continue the desired sequence. When complete sequence is detected we have to make output HIGH and go to the initial state. This is illustrated in Fig. 8.6.1 (c).

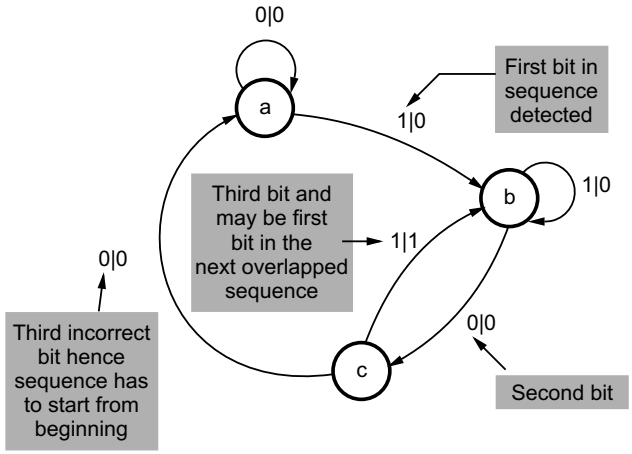


Fig. 8.6.1 (c)

From the above state diagram we can determine the state table as shown in Table 8.6.1 (a).

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
a	a	b	0	0
b	c	b	0	0
c	a	b	0	1

Table 8.6.1 (a) State table

Since there are three states we need two flip-flops. Assigning state a = 00, state b = 01 and state c = 10 we can determine the excitation table as shown in Table 8.6.1 (b).

Present state		Next state		Output Z	
A	B	A^+	B^+		
		X = 0	X = 1	X = 0	X = 1
0	0	0 0	0 1	0	0
0	1	1 0	0 1	0	0
1	0	0 0	0 1	0	1

Table 8.6.1 (b) Excitation table

K-map simplification

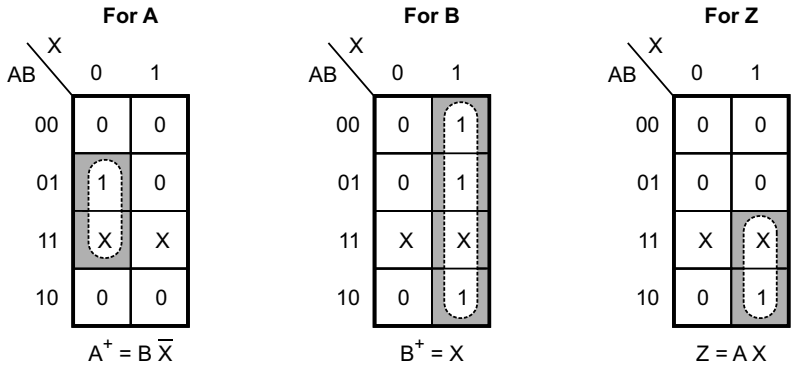


Fig. 8.6.2 (a)

Logic diagram

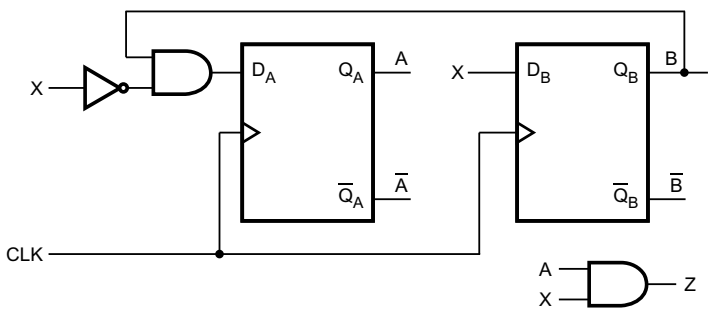


Fig. 8.6.2 (b)

Example 8.6.2 *Design a Moore type sequence detector to detect a serial input sequence of 101.*

Solution : In a Moore type design output depends only on flip-flop states. Therefore, in the state diagram of Moore machine, the output is written with the state instead of with the transition between the states. Apart from this the process of determining the state diagram is similar to that used for Mealy machine. Let us start with state 'a' as an initial state.

1. State a

When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence.

When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0 since we have not yet detected all the bits in the sequence.

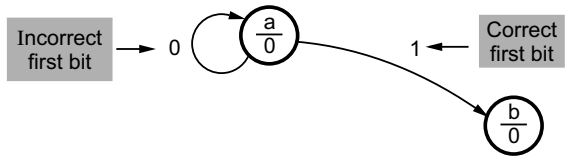


Fig. 8.6.3 (a)

2. State b

When input is 0, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 1, we have to remain in state b because 1 which we have detected may start the sequence. Output is still zero.

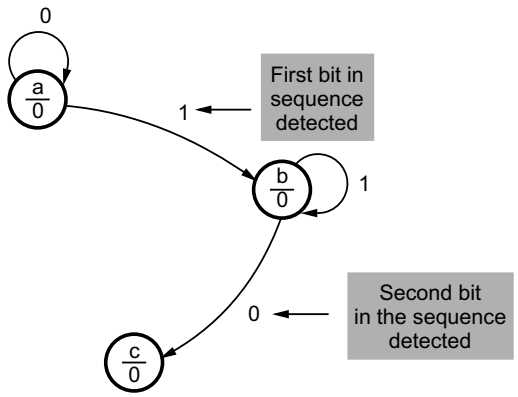


Fig. 8.6.3 (b)

3. State c

Now, when a 1 input occurs, the 101 sequence is completed and output must equal to 1. Here, we cannot go back to state b (since output in state b is zero) and hence we have to create new state d with a output 1.

In case if input is zero, we have to restart checking of input sequence and hence we have to return to state a.

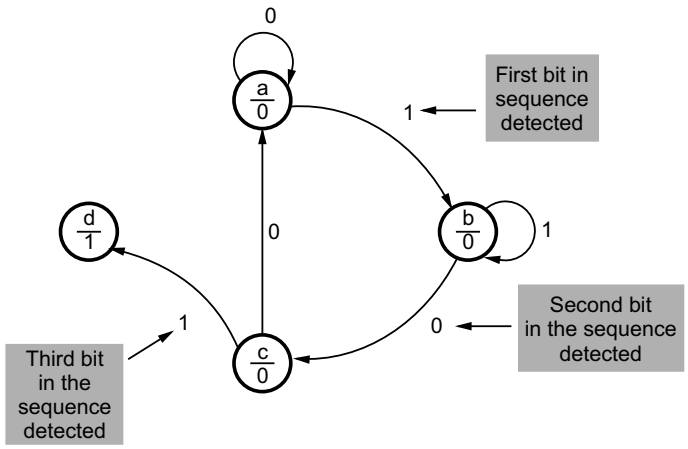


Fig. 8.6.3 (c)

4. State d

Since the sequence is detected, this is the last state. When input is 1, we have detected the first bit in the next sequence, hence we have to go to state b.

When input is 0, we have detected the second bit in the overlapped sequence, hence we have to go to state c.

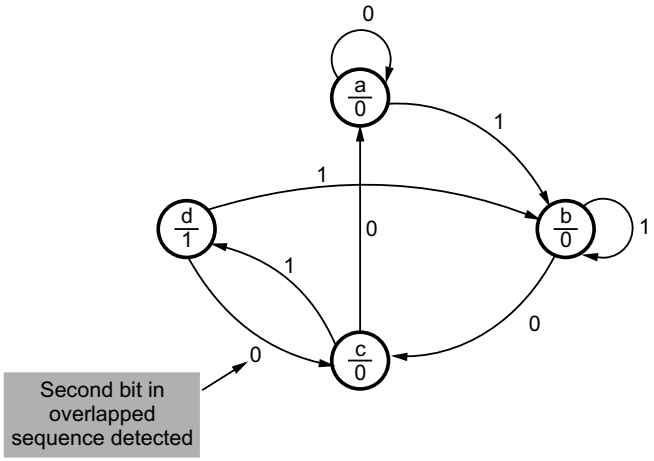


Fig. 8.6.3 (d)

From the above state diagram we can determine the state table as shown in Table 8.6.2 (a).

Present state	Next state		Output Z
	X = 0	X = 1	
a	a	b	0
b	c	b	0
c	a	d	0
d	c	b	1

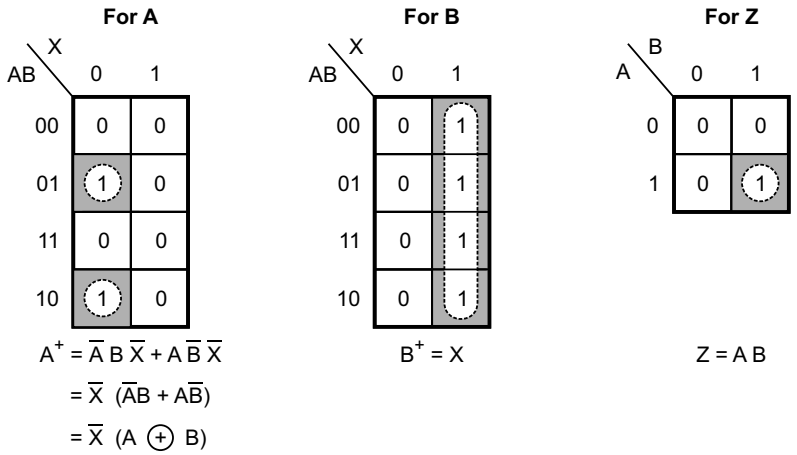
Table 8.6.2 (a) State table

Since there are four states we need two flip-flops. Assigning state a = 00, b = 01, c = 10 and d = 11 we can determine the excitation table as shown in Table 8.6.2 (b).

Present state		Next state A^+ B^+				Output
A	B	X = 0		X = 1		Z
0	0	0	0	0	1	0
0	1	1	0	0	1	0
1	0	0	0	1	1	0
1	1	1	0	0	1	1

Table 8.6.2 (b) Excitation table

K-map simplification



$A^+ = \overline{A}\,B\,\overline{X} + A\,\overline{B}\,\overline{X}$

$= \overline{X}\,(\overline{A}B + A\overline{B})$

$= \overline{X}\,(A \oplus B)$

$B^+ = X$

$Z = A\,B$

Fig. 8.6.4

Logic diagram

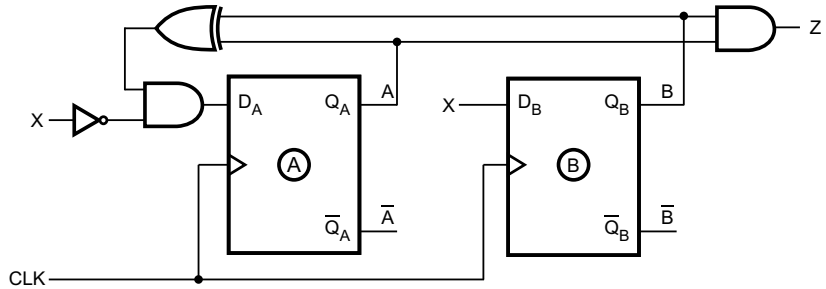


Fig. 8.6.5

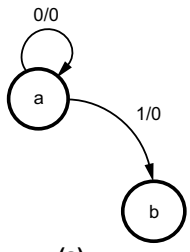
Examples with Solutions

Example 8.6.3 *Design a sequence detector to detect the following sequence using JK flip-flops. (Use Mealy Machine) … 110*

Solution : The given sequence has 3-bit, so we require 3 states in the state diagram. Let us start drawing direction lines, assuming state 'a' is an initial state.

State a :

When input is 1, we have detected the first bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to remain in the state 'a' because bit 0 is not the first bit in the sequence. In both cases output is 0, since we have not yet detected all the bits in the sequence.



(a)

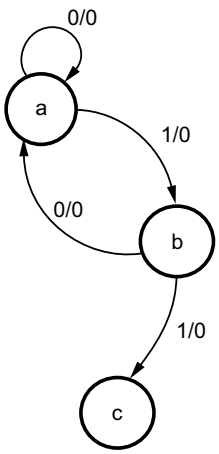
State b :

When input is 1, we have detected the second bit in the sequence, hence we have to go to the next state to detect the next bit in the sequence. When input is 0, we have to go to the state 'a' to detect the first bit in the sequence, i.e. 1.

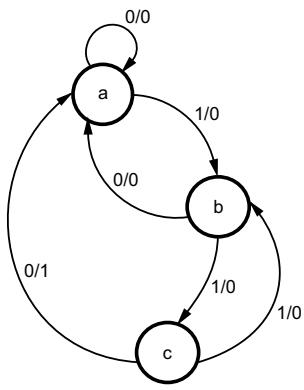
State c :

When input is 0, we have detected the last bit in the sequence, hence we have to go to the initial state, to detect the next sequence and make output high indicating sequence is detected. When input is 1, we have to go to the state 'b' because 1 which we have detected may start the sequence.

Assuming state assignments as a = 00, b = 01 and c = 10, we can determine excitation table for above state diagram as shown in Table 8.6.3.



(b)



(c)

Fig. 8.6.6 State diagram

Input	Present state		Next state		Output	Flip-flop inputs			
X	A_n	B_n	A_{n+1}	B_{n+1}	Y	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	1	0	0	0	0	X	X	1
0	1	0	0	0	1	X	1	0	X
1	0	0	0	1	0	0	X	1	X
1	0	1	1	0	0	1	X	X	1
1	1	0	0	1	0	X	1	1	X

Table 8.6.3 Excitation table

K-map simplification

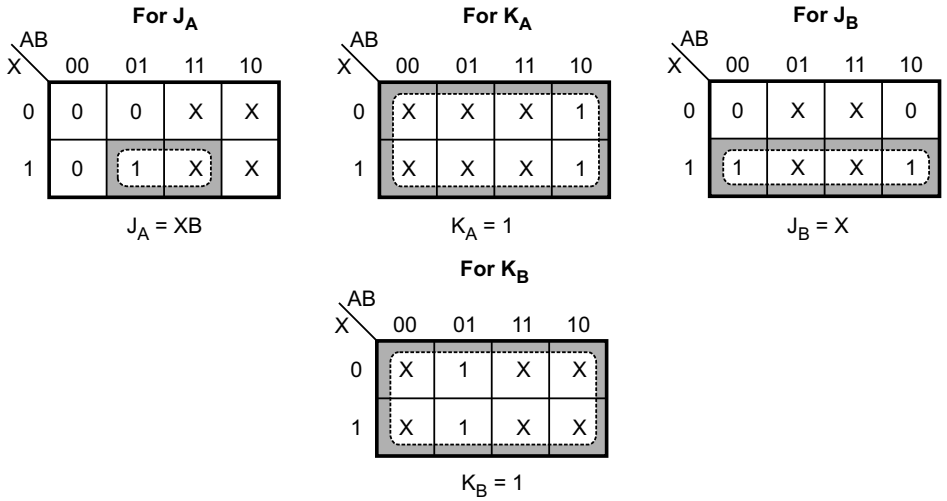


Fig. 8.6.7

Logic diagram

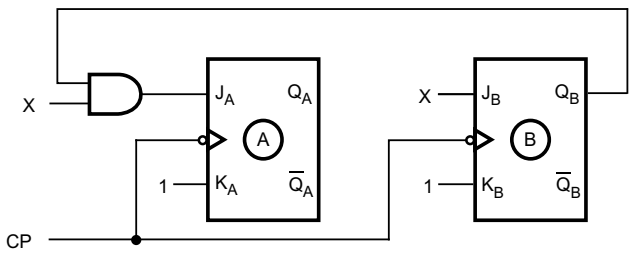


Fig. 8.6.8

Example 8.6.4 *Design a sequence detector to detect sequence 1101.*

Solution : State diagram

In Moore model, output depends only on the present state and not on the input, so state diagram is as shown in the Fig. 8.6.9.

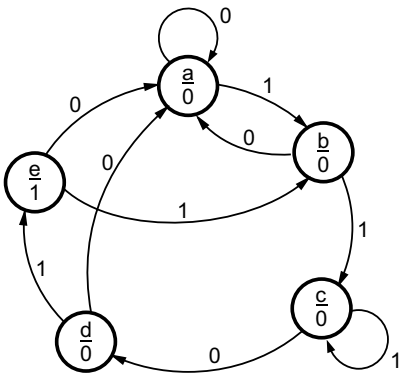


Fig. 8.6.9

State table

Present state	Next state		Output
	X = 0	X = 1	Y
a	a	b	0
b	a	c	0
c	d	c	0
d	a	e	0
e	a	b	1

Table 8.6.4

State assignment

a = 000, b = 001, c = 010, d = 011 and e = 100

State transition table

Present state			Input	Next state			Output
Q_A	Q_B	Q_C	X	Q_{A+}	Q_{B+}	Q_{C+}	Y
0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0
0	0	1	1	0	1	0	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	0	0	0
0	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	1	1
1	0	1	0	×	×	×	×
1	0	1	1	×	×	×	×
1	1	0	0	×	×	×	×
1	1	0	1	×	×	×	×
1	1	1	0	×	×	×	×
1	1	1	1	×	×	×	×

Table 8.6.5

K-map simplification

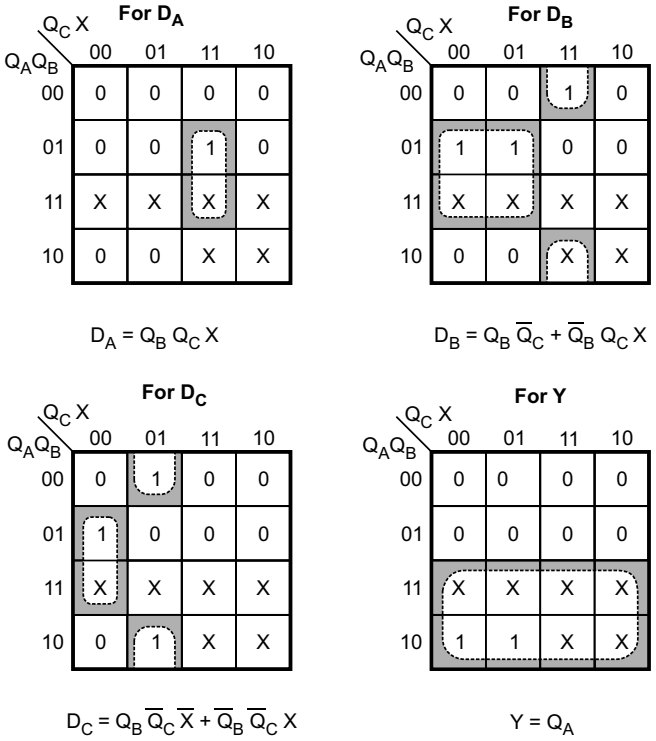

For D_A
Q_C X
$Q_A Q_B$
00 01 11 10
00 0 0 0 0
01 0 0 1 0
11 X X X X
10 0 0 X X
$D_A = Q_B Q_C X$
For D_B
Q_C X
$Q_A Q_B$
00 01 11 10
00 0 0 1 0
01 1 1 0 0
11 X X X X
10 0 0 X X
$D_B = Q_B \overline{Q}_C + \overline{Q}_B Q_C X$
For D_C
Q_C X
$Q_A Q_B$
00 01 11 10
00 0 1 0 0
01 1 0 0 0
11 X X X X
10 0 1 X X
$D_C = Q_B \overline{Q}_C \overline{X} + \overline{Q}_B \overline{Q}_C X$
For Y
Q_C X
$Q_A Q_B$
00 01 11 10
00 0 0 0 0
01 0 0 0 0
11 X X X X
10 1 1 X X
$Y = Q_A$

Fig. 8.6.10

Logic diagram

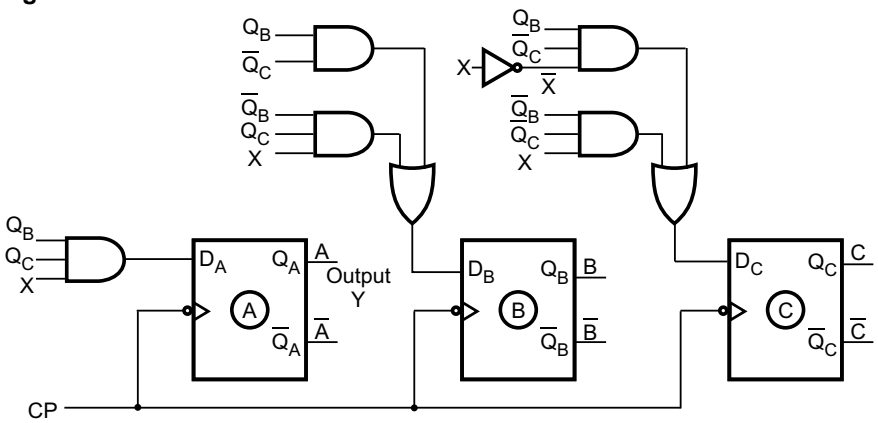


Fig. 8.6.11

Example for Practice

Example 8.6.5 *Design a sequence detector which detects the sequence "01110" using D flipflops (one bit overlapping).* **Dec.-12, Marks 8**

8.7 Simplification of Incompletely Specified Machines

May-12

We often encountered conditions in designing combinational logic circuits where the truth table was incompletely specified, which resulted in don't-care terms. The same thing can happen in sequential circuits if either state transitions or output variables are not completely specified. Such machines are said to be incompletely specified. In this section we see the determination of properties and methods for simplifying the incompletely specified machines.

When the state transitions are not specified, the sequential machine is not predictable. It is desirable to avoid such situations either by specifying inputs sequences so that no next state is unspecified or by assigning next states that are not contrary to the desired result. This is illustrated in Table 8.7.1 (a) and Table 8.7.1 (b). Table 8.7.1 (a) gives the incomplete description of a sequential machine. However, Table 8.7.1 (b) adds state T to describe specified behaviour of sequential machine, in which all state transitions are specified, and only the outputs are partially defined.

PS	NS, Z	
	X = 0	X = 1
A	B, 0	–
B	–, 0	C, 1
C	A, 0	B, 0

Table 8.7.1 (a)

PS	NS, Z	
	X = 0	X = 1
A	B, 0	T, –
B	T, 0	C, 1
C	A, 0	B, 0
T	T, –	T, –

Table 8.7.1 (b)

Unspecified outputs can be assigned without any impact on the state sequence. However, it is advantageous to leave outputs unspecified as long as possible during the state reduction process. This provides additional flexibility in the state reduction process.

When reducing incompletely specified state table we use term **state compatibility** instead of **state equivalence**. The state compatibility is defined as :

States s_i and s_j said to be compatible states, if and only if for every input sequence that affects the two states, the same output sequence occurs whenever both outputs are specified and regardless of whether s_i or s_j is the initial state.

Consider the state table shown in Table 8.7.2. In this table there are following compatible states.

1. $s_0 = s_4$, if the output z, for s_4 when X = 0 and X = 1, is changed from a don't care to a 0.
2. $s_1 = s_3$, if the output Z, for s_1 and s_3 when X = 0, is changed from a don't care to a either a 0 or a 1.

Note : s_0 and s_4 are compatible.

PS	NS, z	
	X = 0	X = 1
s_0	s_0, 0	s_1, 0
s_1	s_0, –	s_2, 0
s_2	s_3, –	s_3, 1
s_3	s_4, –	s_2, 0
s_4	s_0, –	s_1, –

Table 8.7.2 Incompletely specified state table

Review Question

1. *With an example, explain the use of algorithmic state machines.* **May-12**

8.8 Algorithmic State Machine

May-12

The main components of digital system are : Control logic and datapath. The relationship between these two is shown in the Fig. 8.8.1. The data processing path, commonly known as a **datapath**, manipulates data in registers according to system's requirements. The control logic initiates a sequence of commands to the datapath. It uses status conditions from the datapath as decision variables for determining the sequence of control signals.

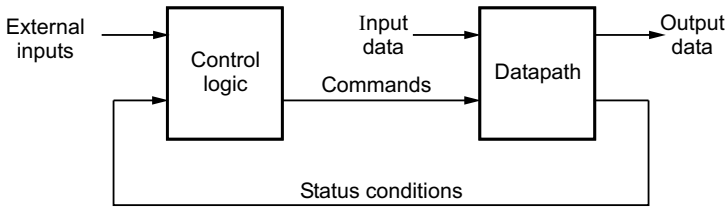


Fig. 8.8.1 Interaction between control logic and datapath

The control sequence and datapath tasks of a digital system are specified by means of a hardware algorithm. A hardware algorithm is a step-by-step procedure to implement the desired task with a selected circuit components. We know that, a flow chart is a convenient way to specify the sequence of procedural steps and decision paths for an algorithm. A special flowchart that has been developed specifically to define digital hardware algorithms is called an **Algorithmic State Machine** (ASM) chart. A state machine is another term used for sequential circuit.

The ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.

8.8.1 ASM Symbols/Notations

An ASM chart consists of three basic elements : the state box, the decision box, and the conditional box.

State box : A state in the control sequence is indicated by a state box, as shown in the Fig. 8.8.2.

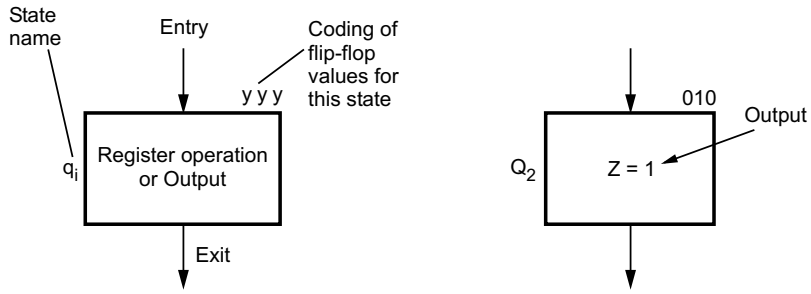


Fig. 8.8.2 State box

As shown in the Fig. 8.8.2, rectangle shape is used to represent state box. (At the left of this box is the name of the state, such as A, B, Q_0, Q_2. On the right hand top corner of the box is a list of the flip-flop values that define that state. The circuit outputs that can occur whenever the circuit is in the corresponding state regardless of input values are listed within the box.) Finally, a line drawn from state box, known as exit, indicates the path to the next state.

Decision box :

The decision box describes the effect of an input on the control subsystem. It has a diamond shape box with two or more exit paths, as shown in the Fig. 8.8.3. The input condition to be tested is written inside the diamond box. Each value either 0 or 1 that may be assumed by an input or expression in a diamond is associated with an exit path from that diamond. These paths lead to the blocks corresponding to the next states of the circuit following the next clock pulse.

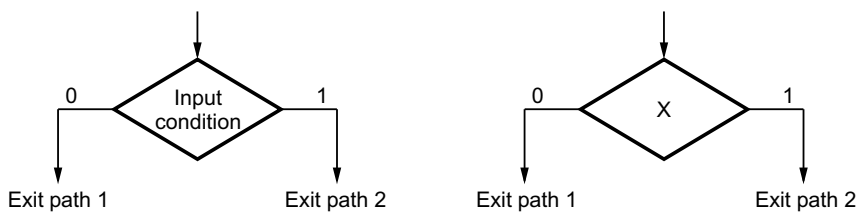


Fig. 8.8.3 Decision box

Conditional box :

The state and decision boxes are familiar from use in conventional flowcharts. The third basic element, the conditional box, is unique to the ASM chart. It is an oval shape box. Its rounded corners differentiate it from the state box. The input path for the conditional box always comes from one of the exit paths of a decision box. The outputs that occur when the path to a conditional output box is satisfied are listed within the box.

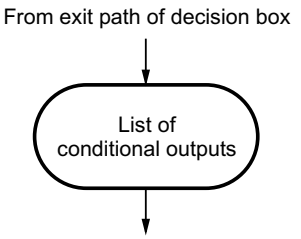


Fig. 8.8.4 Conditional box

The outputs that are no listed in either the state box or a conditional box in a particular ASM chart are always inactive when the circuit is in that state.

Example 8.8.1 *Draw the ASM chart for the following state diagram.* **May-11, Marks 8**

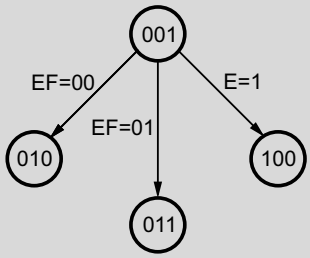


Fig. 8.8.5

Solution :

ASM Block

As shown in the Fig. 8.8.5 (a), an ASM block consists of one state box and all the conditional and decision boxes necessary to determine the path for next state box/boxes. It has one entrance and one or many exit paths. It describes the state of the system during one clock pulse interval.

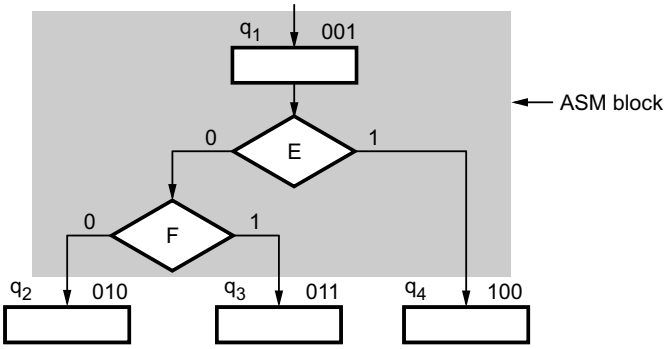


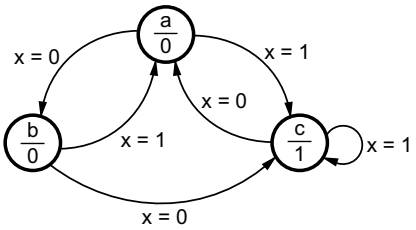
Fig. 8.8.5 (a)

8.8.2 Relation between ASM Chart and State Diagram

Moore Model

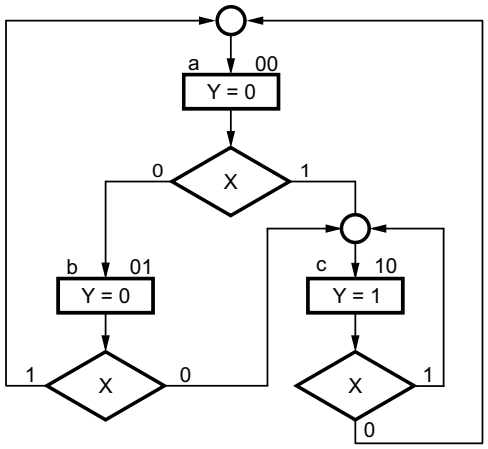
The Fig. 8.8.6 (a) shows the state diagram of Moore machine and the Fig. 8.8.6 (b) shows the ASM chart for it.

- Each state and its assigned binary value is written on state box at left and right corner respectively.
- Since it is a state diagram of Moore machine, the output status is written within the state box.



State	Assignment
a	00
b	01
c	10

(a) State diagram of Moore machine



Note : Y indicates output

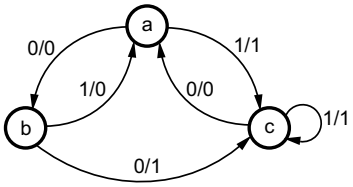
(b) ASM chart for Moore machine

Fig. 8.8.6

- The relation between present state and next state represented by direction lines in the state diagram is shown by decision boxes and corresponding exit paths in the ASM chart.

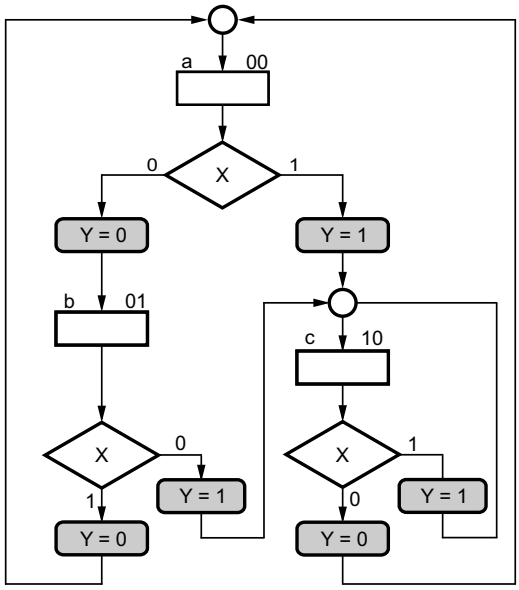
Mealy Model

The Fig. 8.8.7 (a) shows the state diagram of Mealy machine and the Fig. 8.8.7 (b) shows the ASM chart for it.



State	Assignment
a	00
b	01
c	10

(a) State diagram of Mealy machine



(b) ASM chart for Mealy machine

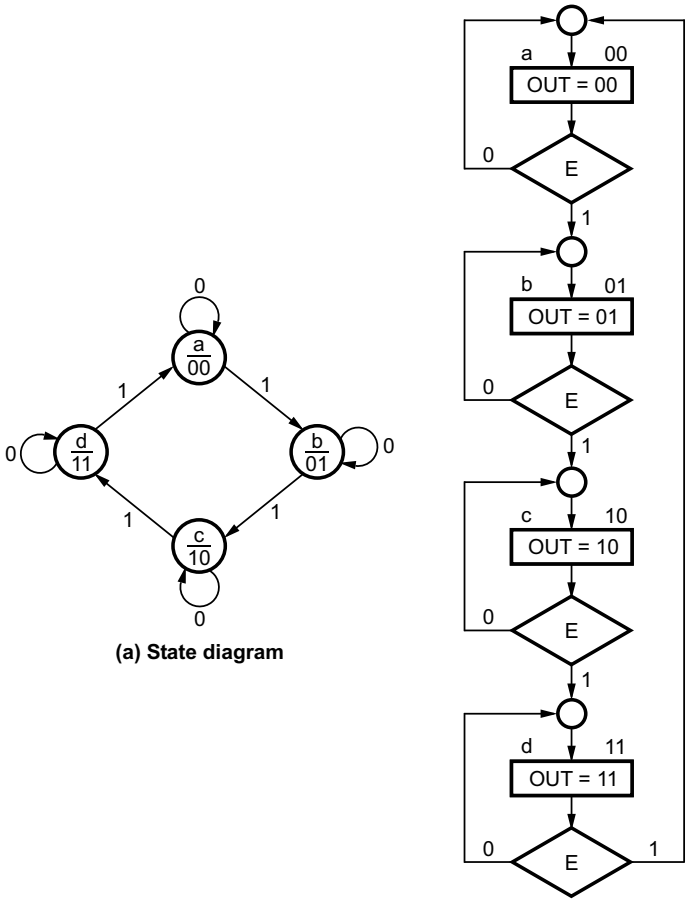
Fig. 8.8.7

The ASM chart for Mealy machine is similar to ASM chart for Moore machine except the representation of output state. We know that in Mealy circuits the output depends on both the present state and the input(s). Due to the reason, here, the output is not specified in the state box; it is specified in the conditional box, as shown in the Fig. 8.8.7 (b). (See Fig. 8.8.7 on previous page.)

Examples with Solutions

Example 8.8.2 *Draw an ASM chart and state diagram for a 2-bit binary counter having enable line E such that : E = 1 (counting enabled) E = 0 (hold present count).*

Solution :



(a) State diagram

(b) ASM chart

Fig. 8.8.8

The output represents the output of the counter.

Example 8.8.3 *Draw an ASM chart and state diagram for 2-bit up/down counter having a mode control input M. For M = 1 : Count up and M = 0 : Count down.*

Solution :

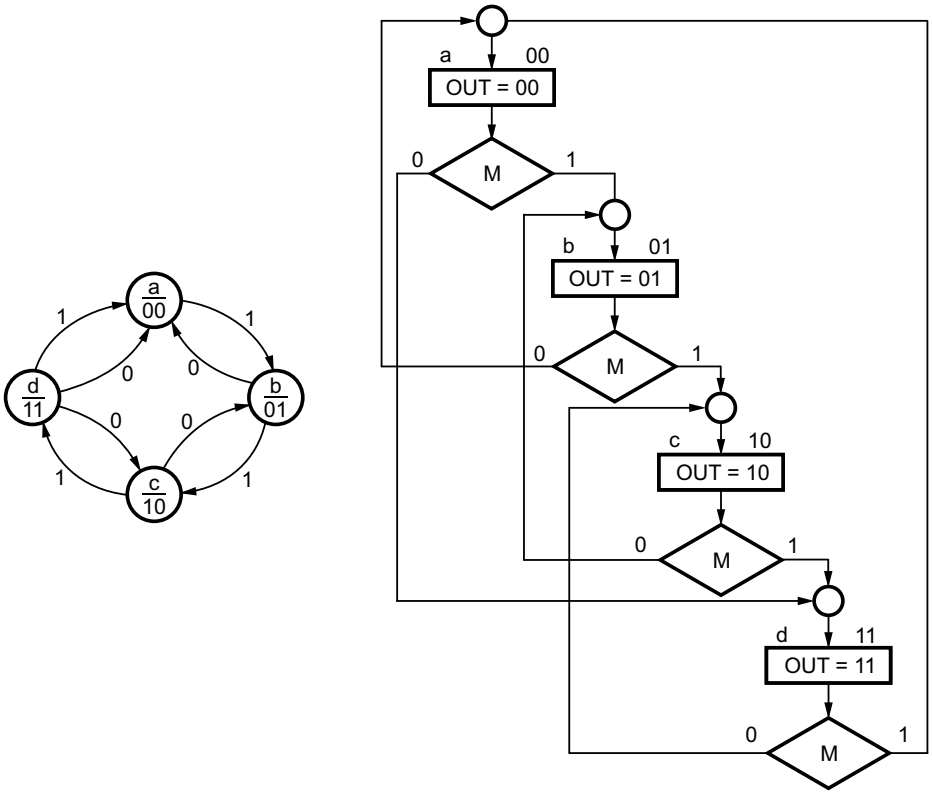


Fig. 8.8.9

Example 8.8.4 *Construct the state table, state diagram and ASM chart for the mealy sequential circuit given in Fig. 8.8.10 (a)*

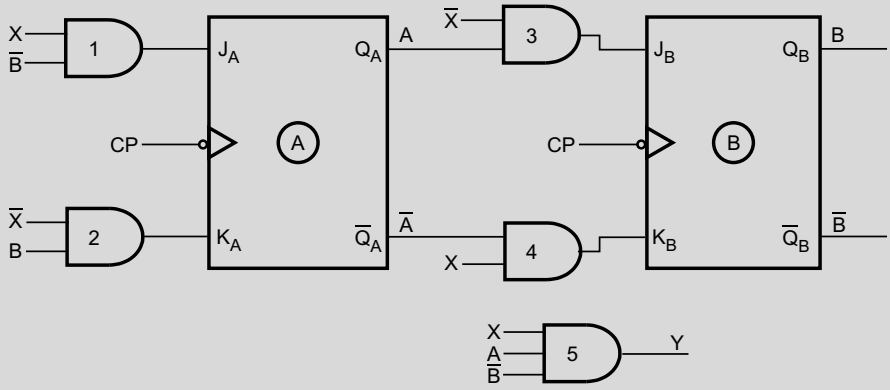


Fig. 8.8.10 (a) Example of a Mealy model

Solution : 1. Determine the flip-flop input equations and the output equations from the sequential circuit.

$$Y = X A \overline{B}$$

$$J_A = X\overline{B} \qquad K_A = \overline{X} B$$

$$J_B = \overline{X} A \qquad K_B = X\overline{A}$$

2. Derive the transition equations.

The transition equations for JK flip-flops can be derived from the characteristic equation of JK flip-flop as follows :

We know that for JK flip-flop,

$$Q^+ = J\overline{Q} + \overline{K}Q$$

$$\therefore \; A^+ = Q_A^+ = J_A \overline{Q}_A + \overline{K}_A Q_A = X\overline{B}\,\overline{Q}_A + \overline{\overline{X}\,B}\,Q_A$$

$$= X\overline{B}\,\overline{A} + \left(X + \overline{B}\right) A$$

and $$B^+ = Q_B^+ = J_B \overline{Q}_B + \overline{K}_B Q_B = \overline{X} A \overline{Q_B} + \overline{X\overline{A}}\; Q_B$$

$$= \overline{X} A \overline{B} + \left(\overline{X} + A\right) B$$

3. Plot a next-state maps for each flip-flop.

The next-state maps are :

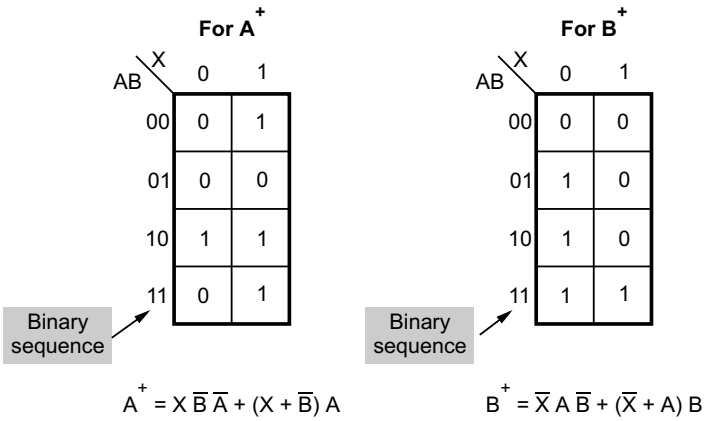


Fig. 8.8.10 (b)

4. Plot the transition table.

The transition table can be formed by combining the above two maps. The Table 8.8.1 (a) shows the transition table.

Present State	Next state		Output	
A B	X = 0	X = 1	$Y = XA\overline{B}$	
	$A^+ B^+$	$A^+ B^+$	X = 0	X = 1
0 0	0 0	1 0	0	0

0 1	0 1	0 0	0	0
1 0	1 1	1 0	0	1
1 1	0 1	1 1	0	0

Table 8.8.1 (a) Transition table

Note : For Mealy sequential circuit output depends on present state as well as on input.

5. Draw the state table.

By assigning a = 00, b = 01, c = 10 and d = 11 we can write state table from the transition table as shown below.

Present state	Next state		Output	
A B	X = 0	X = 1	Y	
	$A^+ B^+$	$A^+ B^+$	X = 0	X = 1
a (00)	a	c	0	0
b (01)	b	a	0	0
c (10)	d	c	0	1
d (11)	b	d	0	0

Table 8.8.1 (b) State table

6. Draw the state diagram

From the state table we can draw the state diagram as shown in Fig. 8.8.11.

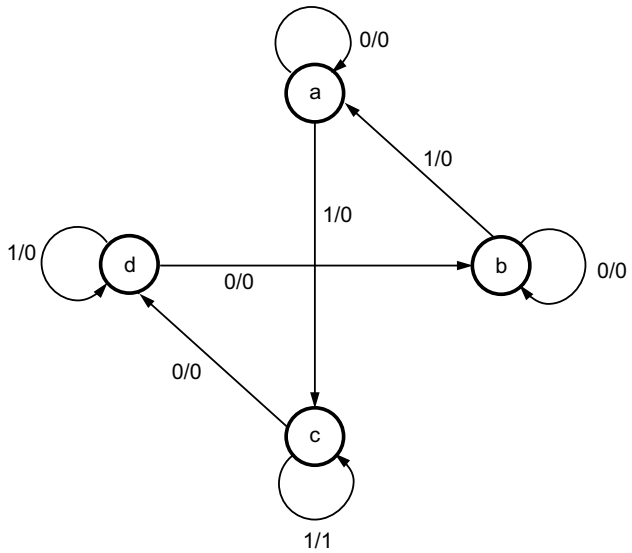


Fig. 8.8.11 State diagram

7. Draw the ASM chart

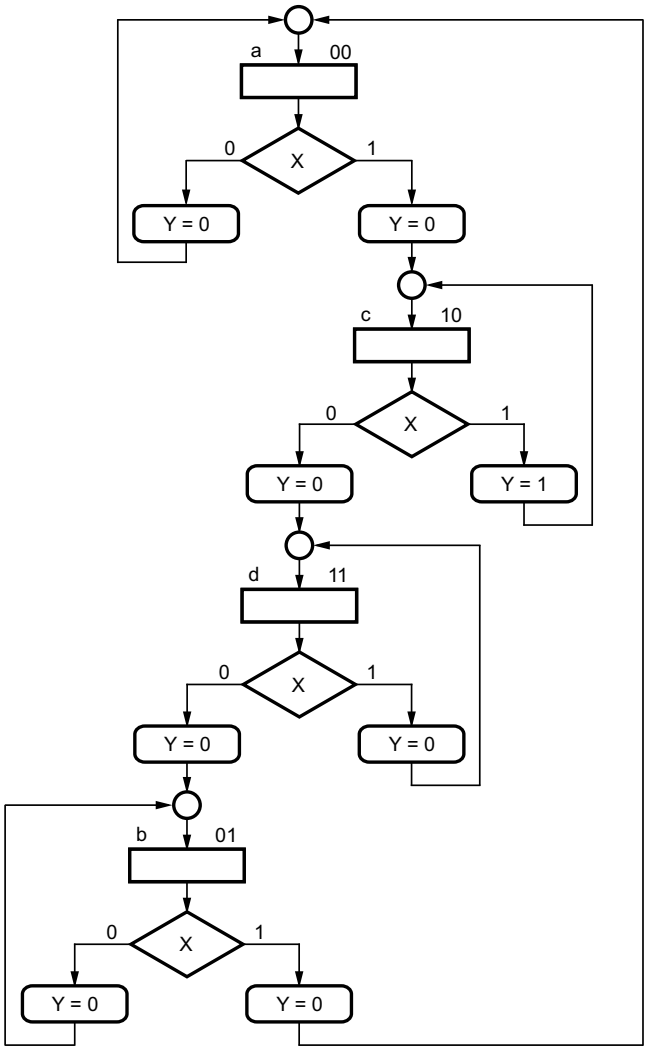


Fig. 8.8.12

Review Questions

1. *Draw and explain the interaction between control logic and datapath.*
2. *Give the notations used in an ASM chart.*
3. *What is ASM block ?*
4. *With an example, explain the use of algorithmic state machines.* **May-12, Marks 8**
5. *Explain the relation between state diagram and ASM chart.*
6. *What are the basic building blocks of a algorithmic state machine chart ?*

8.9 Two Marks Questions with Answers

Q.1 *Give the comparison between synchronous and asynchronous sequential circuits.* **Dec.-10**

Ans. : Refer section 8.1.

Q.2 *Define a state.* **Dec.-06; May-09**

Ans. : The information stored in the memory elements at any given time defines the state at that time of the corresponding sequential circuit.

Q.3 *What are the classification of sequential circuits ?*

Ans. : The sequential circuits are classified on the basis of timing of their signals into two types. They are,

1. Synchronous sequential circuit. 2. Asynchronous sequential circuit.

Q.4 *What do you mean by present state ?*

Ans. : The information stored in the memory elements at any given time defines the present state of the sequential circuit.

Q.5 *What do you mean by next state ?*

Ans. : The present state and the external inputs determine the outputs and the next state of the sequential circuit.

Q.6 *Draw the block diagram of Moore model.* **May-10, 12**

Ans. : Refer section 8.2.1

Q.7 *What is a Mealy machine ? Give an example.* **Dec.-08**

Ans. : Refer section 8.2.2

Q.8 *Compare Moore and Mealy models.* **May-09, Dec.-10**

Ans. : Refer section 8.2.3.

Q.9 *Define state table.* **May-12**

Ans. : For the design of sequential counters we have to relate present states and next states. The table which represents the relationship between present states and next states is called state table.

Q.10 *What are state diagrams and state tables ?* **May-06**

Ans. : State diagram is a pictorial representation of a behaviour of a sequential circuit. The state is represented by the circle, and the transition between states for different input conditions are indicated by directed lines connecting the circles.

State table is the translation of state diagram into a tabular form; representing relationships among input, output and flip-flop states.

Q.11 *Explain about state reduction or Why is state reduction necessary ?* **Dec.-04, 06; May-09**

Ans. : The state reduction is a technique that reduces the number of states in the sequential circuit by keeping only one state for two or more redundant/ equivalent states. This reduces the number of required flip-flops and logic gates, reducing the cost of the final circuit. Two states are said to be redundant or equivalent, if every possible set of inputs generate exactly same output and same next state.

Q.12 *What is lockout ? How it is avoided ?*

Ans. : In a counter, if the next state of some unused state is again some unused state, it may happen that the counter remains in unused states never to arrive at a used state. Such a condition is called **a lockout condition.**

To avoid lockout, the counter should be provided with an additional logic circuitry which will force the counter from an unused state to the next state as initial state.

Q.13 *What is a self starting counter ?* **May-10**

Ans. : In a counter if the next state of some unused state is again an unused state and if by chance the counter happens to find itself in the unused states and never arrived at a used state then the counter is said to be in the lockout conditions. The counter which never goes in lockout condition is called **self starting counter.**

Q.14 *What is Moore machine ?* **Dec.-07**

Ans. : Refer section 8.2.1.

Q.15 *Define state assignment.* **Dec.-05**

Ans. : The state assignment is an one step in the design of sequential circuits which assigns binary values to the states in such a way that it reduces the cost of the combinational circuit that drives the flip-flops.

Q.16 *Compare the ASM chart with a conventional flow chart.*

Ans. : The ASM chart resembles a conventional flow chart, but is interpreted somewhat differently. A conventional flow chart describes the sequence of procedural steps and decision paths for an algorithm without concern for their time relationship. An ASM chart describes the sequence of events as well as the timing relationship between the states of a sequential controller and the events that occur while going from one state to the next.

Q.17 *What are the basic building blocks of an algorithmic state machine chart ?* **May-11**

Ans. : The basic building blocks of an ASM chart are :

- State box
- Decision box and
- Conditional box

Q.18 *Under what circumstances asynchronous circuits are preferred ?* **Dec.-11**

Ans. : Asynchronous circuits can operate faster than synchronous circuits and hence they are preferred when speed is an important criteria.

❑❑❑

9 Asynchronous Sequential Circuits

Syllabus

Design of fundamental mode and pulse mode circuits - Incompletely specified state machines - Problems in asynchronous circuits - Design of hazard free switching circuits.

Contents

9.1 Introduction . . . . . **Dec.-04, 05,** · · · · · · · · · · · · · Marks 2

9.2 Types of Asynchronous Circuits . . . . . **Dec.-03, 06, 12,** · · · · · · · · · · Marks 8

9.3 Analysis of Pulse Mode Asynchronous Circuits

9.4 Design of Pulse Mode Circuit . . . . . **May-08, 09,** · · · · · · · · · · · · Marks 16

9.5 Analysis of Fundamental Mode Sequential Circuits . . . . . **Dec.-10,** · · · · · · · · · · · · · · · · Marks 8

9.6 Design of Fundamental Mode Sequential Circuits . . . . . **May-11, Dec.-08, 10, 11, 12,** · · · · · · Marks 16

9.7 Data Synchronizers

9.8 Mixed Operating Mode Asynchronous. Circuits . . . . . **May-07, Dec.-06,** · · · · · · · · · · · · · · · · Marks 8

9.9 Design of Hazard Free Switching Circuits . . . . . **Dec.-08, 09, 11, 12, May-09, 10, 12** · · · · · · · · · · Marks 16

9.10 Two Marks Questions with Answers

9.1 Introduction

Dec.-04, 05

We have seen, in synchronous sequential circuits, memory elements are clocked flip-flops. In asynchronous sequential circuits, memory elements are either unclocked flip-flops or time delay elements. Therefore, in asynchronous sequential circuits change in input signals can affect memory element at any instant of time. In synchronous circuits, the designer has to consider the time delays involved to determine the maximum operating speed of the clock. In asynchronous circuits, clock is absent and state change occurs according to delay times of the logic. Due to this, asynchronous circuits are more difficult to design. However, because of absence of clock, asynchronous circuits are faster than synchronous circuits.

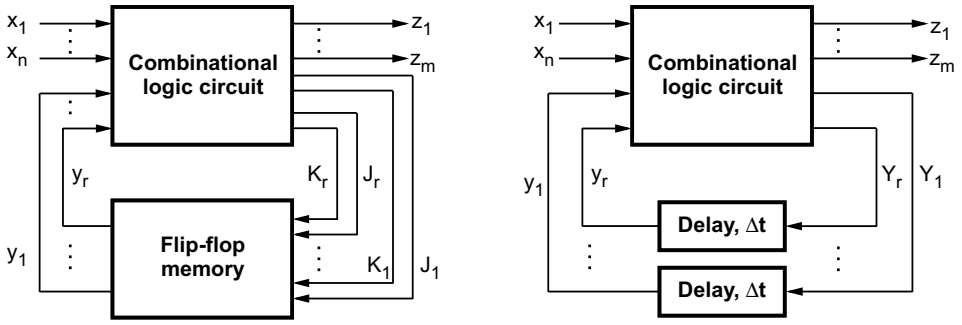
Review Questions

1. *How does the operation of an asynchronous input differ from that of a synchronous input ?* **Dec.-05, Marks 2**
2. *What is an asynchronous sequential circuit ?* **Dec.-04, Marks 2**

9.2 Types of Asynchronous Circuits

Dec.-03, 06, 12

Fig. 9.2.1 shows the block diagram of an asynchronous sequential circuit. It consists of a combinational circuit and delay elements connected to form feedback loops. As shown in the Fig. 9.2.1, there are n input variables, m output variables, and k internal states. The delay elements provide a short term memory for the sequential circuit. The present state and next state variables in asynchronous sequential circuits are called **secondary variables** and **excitation variables**, respectively.



(a) Pulse mode circuit model **(b) Fundamental mode circuit model**

$x_1 - x_n$: n input variables $z_1 - z_m$: m output variables

$J_1 - J_r$, $K_1 - K_r$, $Y_1 - Y_r$: r excitation variables $y_1 - y_r$: r secondary variables

Fig. 9.2.1 Asynchronous sequential circuits

When an input variable changes in value, the secondary variables, i.e. y_1, y_2, …, y_r do not change instantaneously. Certain amount of time is required for the input signal to propagate from the input terminals through the combinational circuit and the delay elements. The combinational circuit generates Y excitation variables which gives the next state of the circuit. The excitation variables are propagated through delay elements to become the new present state for the secondary variables, i.e. y_1, y_2, …, y_r. In the steady state condition excitation and secondary variables are same, but during transition they are different. In other words we can say that, for a given value of input variables, the system is stable if the circuit reaches a steady state condition with $y_i = Y_i$ for $i = 1, 2, \cdots, r$; otherwise the circuit is in a continuous transition and is said to be unstable.

To ensure proper operation, it is necessary for asynchronous sequential circuit to attain a stable state before the input is changed to a new value. Because of unequal delays in the wires and gate circuits, it is impossible to have two or more input variables change at exactly same instant. Therefore, simultaneous changes of two or more input variables are usually avoided. In other words, we can say that only one input variable is allowed to change at any one time and the time between two input changes is kept longer than the time it takes the circuit to reach a stable state.

According to how input variables are to be considered, there are two types of asynchronous circuits :

- Fundamental mode circuits and
- Pulse mode circuits.

Fundamental mode circuit

It assumes that :

- Input changes should be spaced in time by at least Δt, the time needed for the circuit to settle into a stable state following an input change. That is, the input variables should change only when the circuit is stable.
- Only one input variable can change at a given instant of time and
- Inputs are levels and not pulses.
- Delay lines are used as memory elements.

Pulse mode circuit

It assumes that :

- The input variables are pulses instead of levels.
- The width of the pulses is long enough for the circuit to respond to the input.
- The pulse width must not be so long that it is still present after the new state is reached.
- Pulses should not occur simultaneously on two or more input lines.

- Flip-flops are commonly used as a memory elements.
- Memory element transitions are initiated only by input pulses.
- Input variables are used only in the uncomplemented or the complemented forms, but not both.

Review Questions

1. *What is a fundamental mode asynchronous sequential circuit ?* **Dec.-03, Marks 2**
2. *Illustrate pulse mode asynchronous circuit.* **Dec.-06, Marks 8**
3. *Differentiate fundamental mode and pulse mode asynchronous sequential circuits.* **Dec.-12, Marks 10**

9.3 Analysis of Pulse Mode Asynchronous Circuits

In the analysis of pulse mode asynchronous sequential circuits, circuits respond immediately to pulse on their inputs, rather than waiting for clock signal, as in synchronous sequential circuits. The pulse mode circuits assume that pulses do not occur simultaneously on two or more input lines, means that a circuit with n input lines has only n + 1 input conditions, rather than 2^n, as is the case for synchronous circuits. They also assume that a state transition can occur only if an input pulse occurs. Hence, the memory elements of the circuit respond only when an input pulse arrives. Keeping these assumptions in mind, let us examine the behaviour of the pulsed asynchronous circuit in the example 9.3.1.

Examples for Understanding

Example 9.3.1 *Analysis the given pulsed asynchronous sequential circuit.*

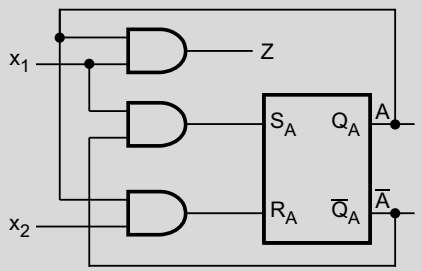


Fig. 9.3.1 A pulsed asynchronous sequential circuit

Solution :

Step 1 : Determine the circuit excitation and output equations.

For given circuit excitation and output equations are :

$$S_A = x_1 \overline{A}$$

$$R_A = x_2 A$$

$$Z = x_1 A$$

Step 2 : Determine the next state equation of state variable.

The characteristic equation for SR flip-flop is

$$Q_n^+ = S + \overline{R} Q_n$$

Using the characteristic equation and excitation equations we have the state variable next state equation is as follows :

$$Q_A^+ = x_1 \overline{A} + (\overline{x_2 A}) A$$

Step 3 : Construct the state variable transition table.

From next state and output equations we can construct the state variable transition table indicating state variables, input variables, next state values and the output value.

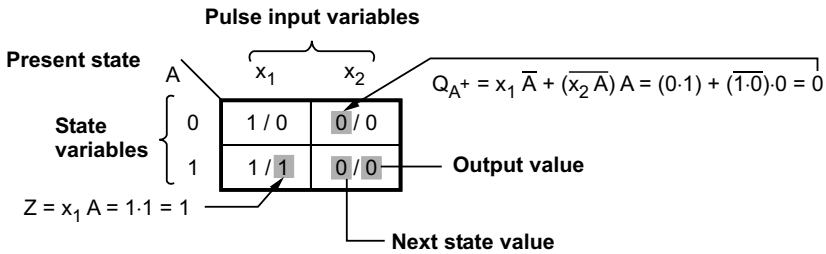
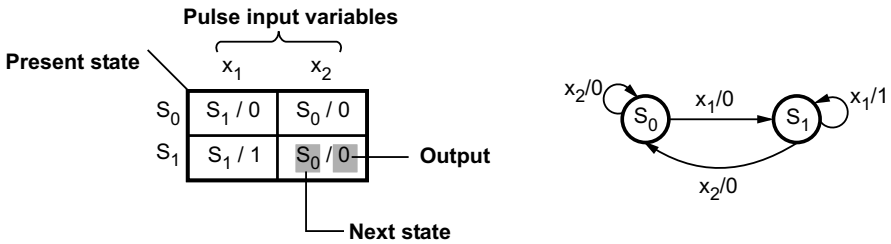


Fig. 9.3.2 State variable transition table

Step 4 : Derive the flow table and state diagram.

From the state variable transition table we can derive the flow table and from flow table we can derive the state diagram as shown in the Fig. 9.3.3. The flow diagram can be constructed by replacing next state and state variable values by actual states $S_0 = 0$ and $S_1 = 1$.



(a) State table (flow table) **(b) State diagram**

Fig. 9.3.3

Step 5 : Draw the timing diagram.

The Fig. 9.3.4 shows the timing diagram for the pulse mode circuit shown in the Fig. 9.3.1. As shown in the timing diagram, the inputs are the pulses and they occur one at a time.

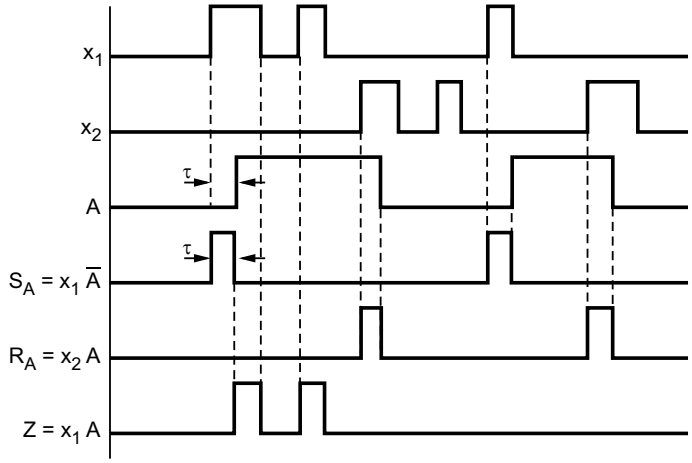


Fig. 9.3.4 Timing diagram for example circuit

Example 9.3.2 *Consider the asynchronous sequential circuit which is driven by the pulses, as shown in the Fig. 9.3.5 (a). Analyze the circuit.*

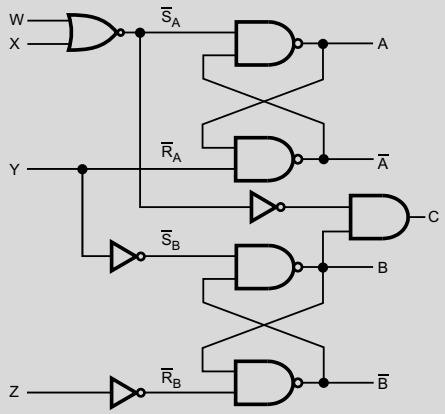


Fig. 9.3.5 Pulse mode asynchronous sequential circuit

Solution : The circuit has two NAND gate latches that generate the state variables, A and B. The circuit has four input variables W, X, Y and Z and one output variable C.

Step 1 : Determine the circuit excitation and output equations.

From the given sequential circuit we can have excitation and output equations as follows :

$$\overline{S}_A = \overline{W + X}\ ,\ S_A = W + X$$

$$\overline{R}_A = Y \qquad R_A = \overline{Y}$$

$$\overline{S}_B = \overline{Y}\ , \qquad S_B = Y$$

$$\overline{R}_B = \overline{Z} \qquad R_B = Z$$

$$C = (W + X) \cdot B$$

Step 2 : Determine next state equations for state variables.

The characteristic equation for SR flip-flop is

$$Q_n^+ = S + \overline{R}\, Q_n$$

Using the characteristic equation and excitation equations we have the state variable next-state equations as follows

$$Q_A^+ = W + X + YA$$

$$Q_B^+ = Y + \overline{Z}\, B$$

Step 3 : Construct the state variable transition table.

From these next state and output equations we can construct the state variable transition table indicating state variables, input variables, next state values and the output-state.

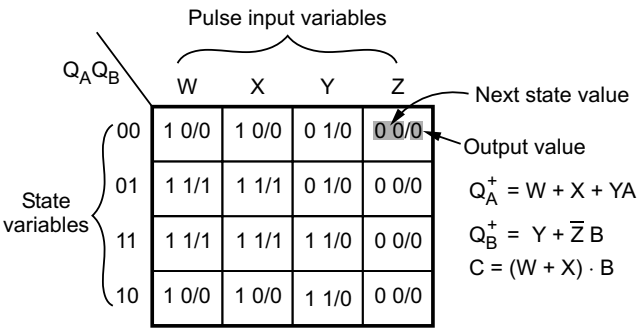
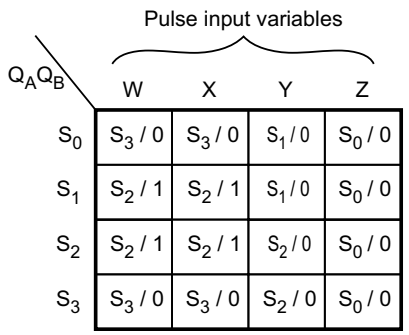
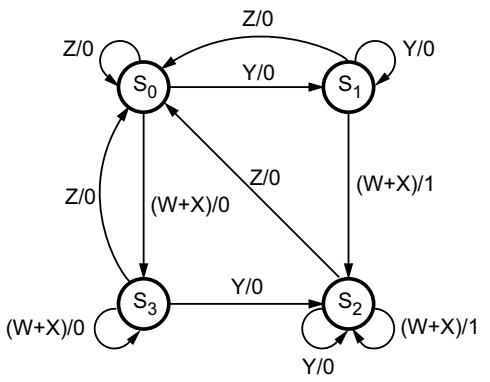


Fig. 9.3.5 (a) State variable transition table

Step 4 : Derive the flow table and state diagram.



(b) State table (Flow diagram)



(c) State diagram

Fig. 9.3.5

From the state variable transition table we can derive the flow table and from flow table we can derive the state diagram as shown in the Fig. 9.3.5 (c) and (d). The flow diagram can be constructed by replacing next state and state variable values by actual states

(S_0 = 00, S_1 = 01, S_2 = 11 and S_3 = 10).

Example 9.3.3 *Consider the asynchronous sequential circuit driven by the pulses shown in Fig. 9.3.6. Analyze the circuit and draw the timing diagram.*

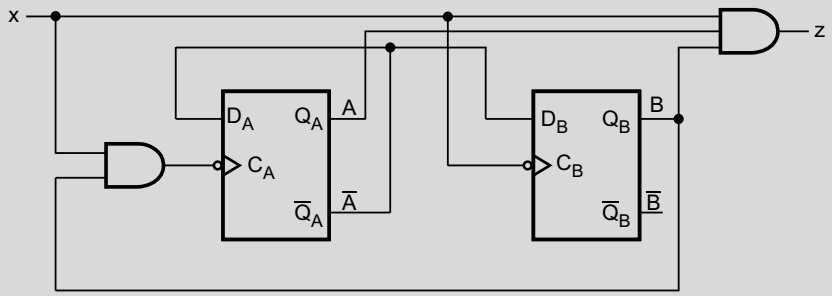


Fig. 9.3.6

Solution :

Step 1 : Determine the circuit excitation and output equations.

From the given sequential circuit we can have excitation and output equations as follows :

$$D_A = \overline{A}, \qquad D_B = \overline{A}$$

$$z = x\ A\ B$$

$$C_A = x\ B \qquad C_B = x$$

Step 2 : Determine the next state equations for state variables.

The characteristic equation for D flip-flop is

$$Q_n^+ = D \text{ and considering clock input it is given as}$$

$$Q_n^+ = DC + Q_n \overline{C}$$

Using the characteristic equation and excitation equations we have the state variable next state equations as follows :

$$Q_A^+ = \overline{A}\ C_A + A\ \overline{C}_A = \overline{A}\ x\ B + A\ (\overline{x\ B}) = \overline{A}\ x\ B + A\ (\overline{x} + \overline{B})$$

$$= \overline{A}\ x\ B + A\ \overline{x} + A\ \overline{B}$$

$$Q_B^+ = \overline{A}\ C_B + B\ \overline{C}_B = \overline{A}x + B\ \overline{x}$$

Step 3 : Construct the state variable transition table.

From these next state and output equations we can construct the state variable transition table indicating state variables, input variables, next state values and the output state.

A state variable transition table may be compiled for this circuit if we define the following.

Input : I_0 = No pulse on x

I_1 = Pulse on x

States : AB : 00, 01, 10, 11

Output : z = 0, z = 1

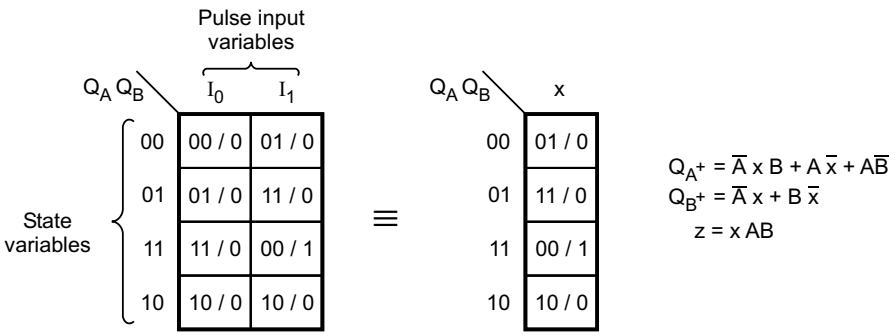
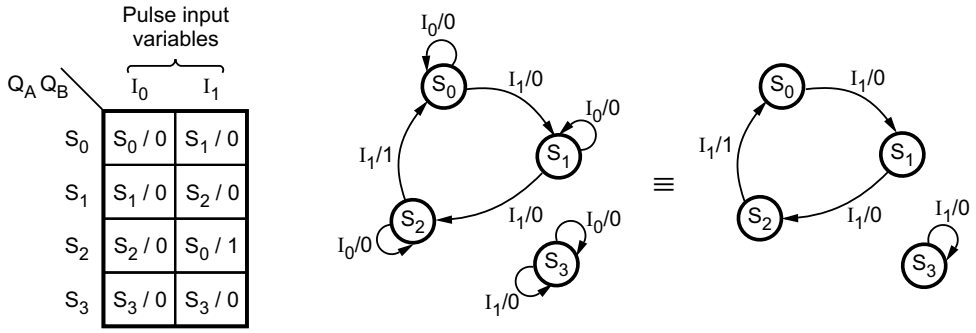


Fig. 9.3.7 State variable transition table

Step 4 : Derive the flow table and state diagram.

From the state variable transition table we can derive the flow table and from flow table we can derive the state diagram as shown in the Fig. 9.3.8. The flow diagram can be constructed by replacing next state and state variable values by actual states

(S_0 = 00, S_1 = 01, S_2 = 11 and S_3 = 10).



(a) State table (Flow table) **(b) State diagram**

Fig. 9.3.8

Step 5 : Draw the timing diagram.

The Fig. 9.3.9 shows the timing diagram for the given circuit.

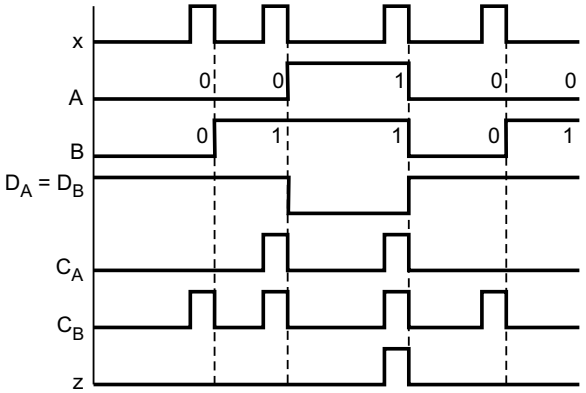


Fig. 9.3.9 Timing diagram

Example for Practice

Example 9.3.4 *Analyze the pulse-mode circuit shown in Fig. 9.3.10. Determine flow table and state diagram.*

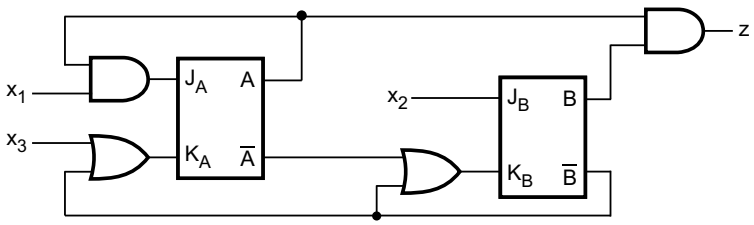


Fig. 9.3.10

Review Question

1. *Describe the procedure of analysis of pulse mode asynchronous circuit with the help of example.*

9.4 Design of Pulse Mode Circuit

May-08, 09

The design of pulse-mode circuits similar to the design of synchronous circuits discussed in Chapter 4. However, when designing pulse-mode circuits, remember that no clock pulse is present, inputs occur on only one line at a time and only uncomplemented forms of input signals may be used.

The absence of a clock pulse indicates that latch or flip-flop triggering must be accomplished by utilizing the pulses on the input signals and therefore all circuit timing information must be obtained from the input pulses. Hence, the input pulses not only

provide input information but also assume the functions performed by the clock pulse in synchronous circuits.

The steps involved in the design of pulse-mode asynchronous sequential circuits are :

1. Define states and draw a state diagram and/or state table of the circuit.
2. Minimize the state table.
3. Do state assignment.
4. Choose the type of latch or flip-flop to be used and determine excitation equations.
5. Construct excitation table for the circuit.
6. Determine the output equation and the flip-flop input equations using k-map simplification.
7. Draw the logic diagram.

Examples for Understanding

Example 9.4.1 *Design a pulse-mode circuit having two input lines, x_1 and x_2 and one output line, z, as shown in Fig. 9.4.1. The circuit should produce an output pulse to coincide with the last input pulse in the sequence x_1 - x_2 - x_2. No other input sequence should produce an output pulse.* **May-08, Marks 16**

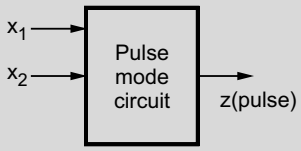


Fig. 9.4.1

Solution :

Step 1 : Define states and draw the state diagram and/or state table of the circuit :

S_0 : indicates that the last input was x_1.

S_1 : indicates that the sequence x_1 - x_2 occurred.

S_2 : indicates that the sequence x_1 - x_2 - x_2 occurred.

The Fig. 9.4.1 (a) shows the state diagram for the given circuit. It is important to note that the format of the state diagram is similar to that used for synchronous circuits. However, the transitions are labeled with the input variable and the output value rather than with both input and output values. Also, remember that the state transitions are triggered by the occurrence of the indicated input pulse and not by a clock pulse.

The state table corresponding to the state diagram of Fig. 9.4.1 (a) is as follows :

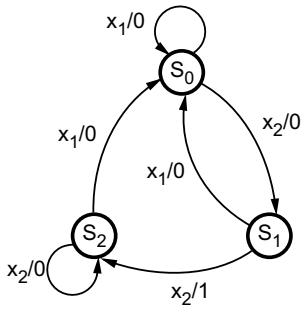


Fig. 9.4.1 (a) State diagram

Present state	Next state, Output Z	
	x_1	x_2
S_0	S_0 / 0	S_1 / 0
S_1	S_0 / 0	S_2 / 1
S_2	S_0 / 0	S_2 / 0

Table 9.4.1 State table

Step 2 : Minimize state table : State table is minimum.

Step 3 : Assign states : A state assignment of $S_0 = 00$, $S_1 = 01$ and $S_2 = 10$.

Step 4 : Flip-flops to be used : T

Note : For T flip-flop output changes when T = 1.

Step 5 : Construct execution table for the circuit.

Present state	Next state		Flip-flop inputs				Output z	
			T_A		T_B			
A B	X_1 A^+ B^+	X_2 A^+ B^+	X_1 $(A \oplus A^+)$	X_2 $(A \oplus A^+)$	X_1 $(A \oplus A^+)$	X_2 $(A \oplus A^+)$	X_1	X_2
0 0	0 0	0 1	0	0	0	1	0	0
0 1	0 0	1 0	0	1	1	1	0	1
1 0	0 0	1 0	1	0	0	0	0	0

Table 9.4.2 Excitation table

Step 6 : K-map simplification for T inputs and Z output.

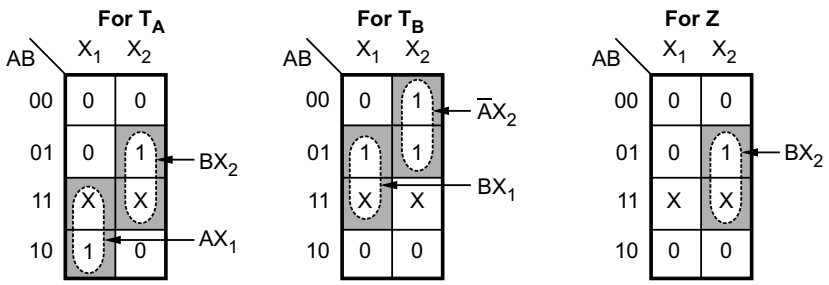


Fig. 9.4.1 (b)

Note : Only vertical grouping is allowed.

$$T_A = AX_1 + BX_2 \qquad T_B = BX_1 + \overline{A}\,X_2 \qquad Z = BX_2$$

Step 7 : Draw the logic diagram.

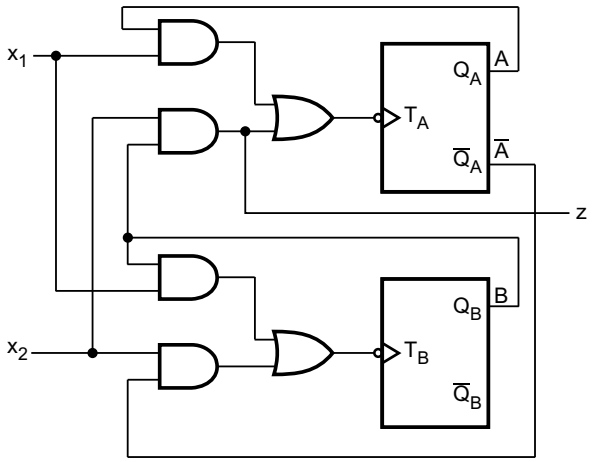


Fig. 9.4.1 (c) Logic diagram for the pulse-mode example

In the previous example, the circuit realization took the form of a Mealy-type circuit since the output was a function of both an input and a state variable. A next example, will now be presented that describes the realization of a Moore-type circuit. Recall that Mealy- and Moore-type circuits were defined in Chapter 8.

Example 9.4.2 *Design a pulse mode circuit with inputs x_1, x_2, x_3 and output z as shown in Fig. 9.4.2. The output should change from 0 to 1, only for input sequence $x_1 - x_2 - x_3$ occurs while z = 0. Also the output z should remain in 1 until x_2 occurs. Use SR flip-flops for the design.* **June-09, Marks 16**

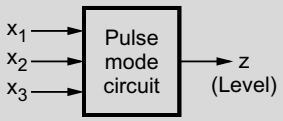


Fig. 9.4.2

Solution :

Step 1 : Draw the state diagram and state table.

Since the output must remain high between input pulses, a Moore-type circuit is required to realize the circuit in Fig. 9.4.2. The state diagram and state table in Fig. 9.4.2 (a) and Table 9.4.3 respectively, satisfy the stated requirements.

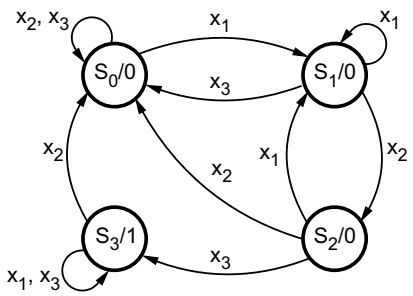


Fig. 9.4.2 (a) State diagram

Present state	Next state			Output Z
	x_1	x_2	x_3	
S_0	S_1	S_0	S_0	0
S_1	S_1	S_2	S_0	0
S_2	S_1	S_0	S_3	0
S_3	S_3	S_0	S_3	1

Table 9.4.3 State table

Step 2 : Minimum state table : State table is minimum

Step 3 : Assign states : A state assignment of $S_0 = 00$, $S_1 = 01$, $S_2 = 11$ and $S_3 = 10$

Step 4 : Flip-flops to be used : SR

Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Excitation table for SR flip-flop

Step 5 : Construct excitation table for the circuit.

Present state		Next state						Z	Flip-flop inputs											
		x_1		x_2		x_3			S_A			R_A			S_B			R_B		
A	B	A^+	B^+	A^+	B^+	A^+	B^+		x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3	x_1	x_2	x_3
0	0	0	1	0	0	0	0	0	0	0	0	X	X	X	1	0	0	0	X	X
0	1	0	1	1	1	0	0	0	0	1	0	X	0	X	X	X	0	0	0	1
1	1	0	1	0	0	1	0	0	0	0	X	1	1	0	X	0	0	0	1	1
1	0	1	0	0	0	1	0	1	X	0	X	0	1	0	0	0	0	X	X	X

Table 9.4.4

Step 6 : K-map simplification for SR inputs and Z output.

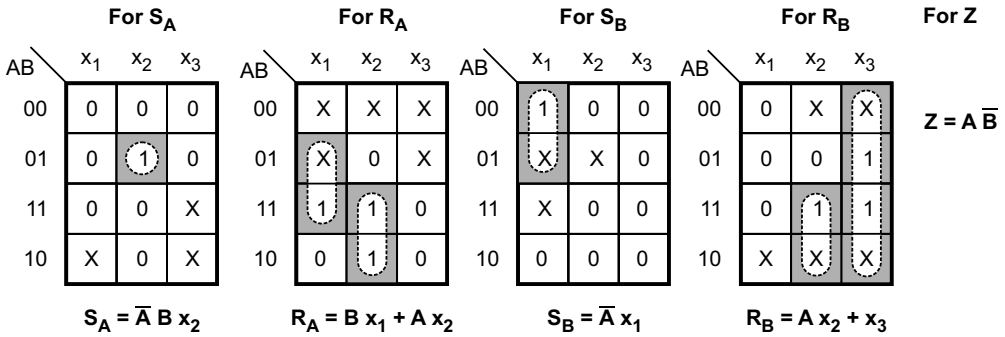


Fig. 9.4.2 (b)

Step 7 : Draw the logic diagram.

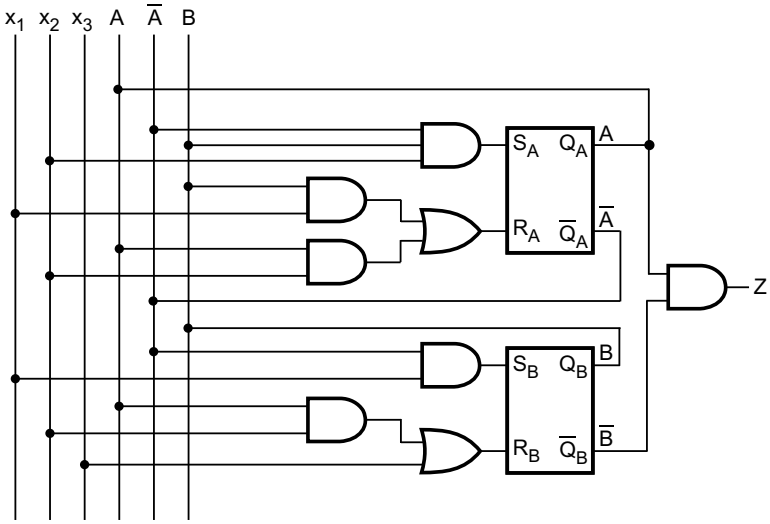


Fig. 9.4.2 (c) Logic diagram for the example

Example for Practice

Example 9.4.3 *A pulse mode asynchronous sequential circuit has two inputs x_1 and x_2, and one output Z. An output transition from 0 to 1 occurs only when the X_2 pulse occurs in the sequence $X_1 - X_2 - X_1 - X_2$. The output Z resets from 1 to 0 only by the first X_1 pulse that occurs following the 0 to 1 output transition. Allow overlapping sequences. Design the circuit using T flip-flops.*

Review Question

1. *List the steps involved in the design of pulse-mode asynchronous circuits.*

9.5 Analysis of Fundamental Mode Sequential Circuits

Dec.-10

Fundamental mode asynchronous sequential circuit analysis requires careful attention because these circuits utilize unclocked memory and level inputs. The procedure to analyze these circuits is as follows :

1. Determine the next-secondary state and output equations from given sequential circuit.
2. Construct the state table.
3. Construct the transition table.
4. Construct output map.

Examples for Understanding

Example 9.5.1 *Analyze the fundamental mode asynchronous sequential circuit given in Fig. 9.5.1.*

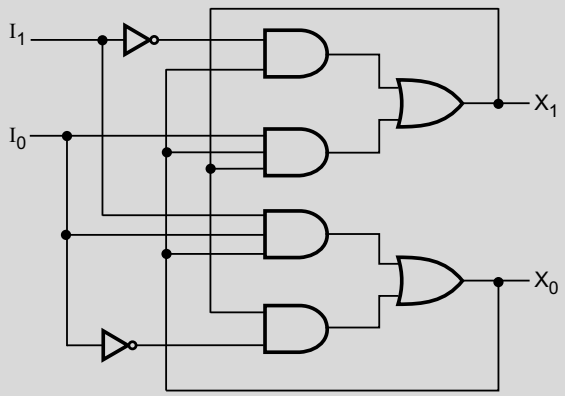


Fig. 9.5.1 Fundamental mode asynchronous sequential circuit

Solution : The given circuit has two input variables I_1 and I_0 and one output variable Z. The circuit has two feedback paths which provide inputs to the gates, creating latching operation necessary to produce a sequential circuit. The feedback path also generates the state variables X_0 and X_1. The next state for the circuit is determined by both, the state of input variables and the state variables.

Step 1 : Determine next-secondary state and output equations.

From the given sequential circuit we can have next-secondary state and output equations as follows

$$X_1^+ = X_0 \bar{I}_1 + X_0 X_1 I_0 \qquad X_0^+ = X_0 I_0 I_1 + X_1 \bar{I}_0 \qquad Z = X_0 X_1 I_0$$

Step 2 : Construct state table.

From these next-secondary state and output equations we can construct the state table indicating present-total state, next-total state, stability of the next-secondary state and the

output. The next-secondary state values are found by assigning present-total state values to the Boolean variables in the next-secondary state equations to determine X_1^+ and X_0^+.

For the given input and secondary state if next-secondary state does not change then the state is said to be stable.

Present total state				Next total state				Stable Total state Yes/No	z
Secondary state		Inputs		Next-secondary state		Inputs			
X_1	X_0	I_1	I_0	X_1	X_0	I_1	I_0		
0	0	0	0	0	0	0	0	Yes	0
0	0	0	1	0	0	0	1	Yes	0
0	0	1	1	0	0	1	1	Yes	0
0	0	1	0	0	0	1	0	Yes	0
0	1	0	0	1	0	0	0	No	0
0	1	0	1	1	0	0	1	No	0
0	1	1	1	0	1	1	1	Yes	0
0	1	1	0	0	0	1	0	No	0
1	1	0	0	1	1	0	0	Yes	0
1	1	0	1	1	0	0	1	No	1
1	1	1	1	1	1	1	1	Yes	1
1	1	1	0	0	1	1	0	No	0
1	0	0	0	0	1	0	0	No	0
1	0	0	1	0	0	0	1	No	0
1	0	1	1	0	0	1	1	No	0
1	0	1	0	0	1	1	0	No	0

Table 9.5.1 State table

Note : The shaded portions show that for given inputs, next-secondary states do not match with the corresponding secondary states and hence they are unstable states.

The Fig. 9.5.2 shows the transition table. The numbers written in the table represent next-secondary state values for

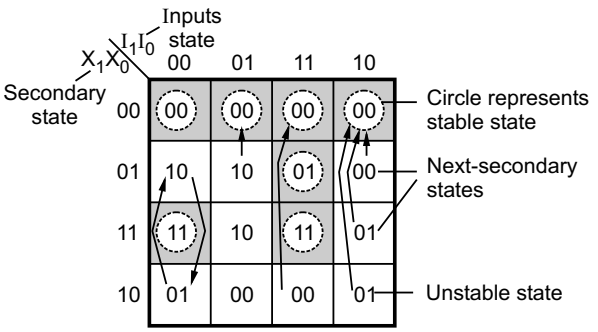


Fig. 9.5.2 Transition table

particular secondary state and inputs. The circle around next-secondary state value indicate that the state is stable. The arrows indicate transitions from unstable states to stable states. For example, if the state (X_1, X_0, I_1, I_0) is 1010, the state value is 01 and it is unstable state. The next-secondary stable state will be 0010 as indicated by arrow. There is no stable state for input $I_1 I_0 = 00$ with secondary states 10 and 01.

X_1X_0 \ I_1I_0	00	01	11	10
00	0	0	0	0
01	–	–	0	–
11	0	–	1	–
10	–	–	–	–

Output is mapped for all stable states. For unstable states output is mapped unspecified.

Fig. 9.5.3 Output map

Example 9.5.2 *An asynchronous sequential circuit is described by the following excitation and output function.*

$Y = X_1X_2 + (X_1 + X_2)Y, \quad Z = Y$

i) Draw the logic diagram of the circuit. ii) Derive the transition table and output map. iii) Describe the behaviour of the circuit.

Solution : i) The logic diagram is as shown in the Fig. 9.5.4 (a).

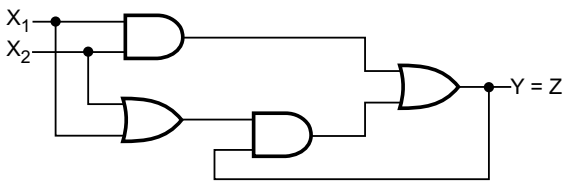


Fig. 9.5.4 (a) Logic diagram

ii)

Present total state			Next total state			Stable total state	Z
S state	Inputs		NS state	Inputs			
Y	X_1	X_2	Y	X_1	X_2	Yes/No	
0	0	0	0	0	0	Yes	0
0	0	1	0	0	1	Yes	0
0	1	1	1	1	1	No	1
0	1	0	0	1	0	Yes	0
1	0	0	0	0	0	No	0
1	0	1	1	0	1	Yes	1
1	1	1	1	1	1	Yes	1
1	1	0	1	1	0	Yes	1

Table 9.5.2 State table

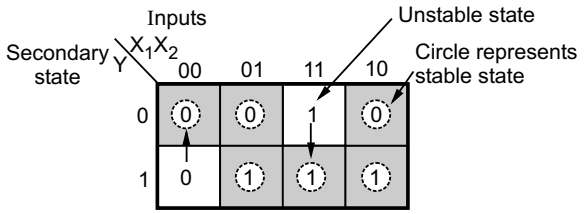


Fig. 9.5.4 (b) Transition table

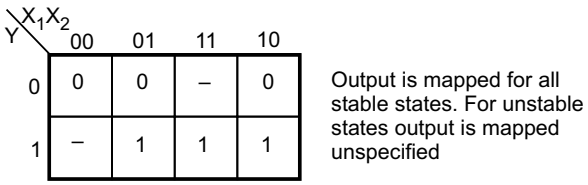


Fig. 9.5.4 (c) Output map

iii) The circuit gives carry output of the full adder circuit.

Example 9.5.3 *An asynchronous sequential circuit has two internal states and one output. The excitation and output function describing the circuit are as follows.*

$Y_1 = x_1x_2 + x_1y_2 + x_2y_1$, $Y_2 = x_2 + x_1y_1y_2 + x_1y_1$, $Z = x_2 + y_1$

Solution : The logic diagram is as shown in the Fig. 9.5.5 (a).

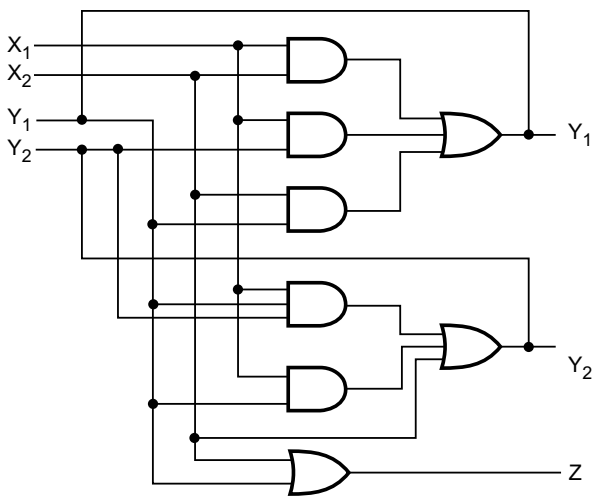


Fig. 9.5.5 (a) Logic diagram for given problem

Present total state				Next total state				Stable total state	Z
S State		Inputs		NS state		Inputs			
Y_1	Y_2	X_1	X_2	Y_1	Y_2	X_1	X_2	Yes/No	
0	0	0	0	0	0	0	0	Yes	0
0	0	0	1	0	1	0	1	No	1
0	0	1	1	1	1	1	1	No	1
0	0	1	0	0	0	1	0	Yes	0
0	1	0	0	0	0	0	0	No	0
0	1	0	1	0	1	0	1	Yes	1
0	1	1	1	1	1	1	1	No	1
0	1	1	0	1	0	1	0	No	0
1	1	0	0	0	0	0	0	No	1
1	1	0	1	1	1	0	1	Yes	1
1	1	1	1	1	1	1	1	Yes	1
1	1	1	0	1	1	1	0	Yes	1
1	0	0	0	0	0	0	0	No	1
1	0	0	1	1	1	0	1	No	1
1	0	1	1	1	1	1	1	No	1
1	0	1	0	0	1	1	0	No	1

Table 9.5.3 State Table

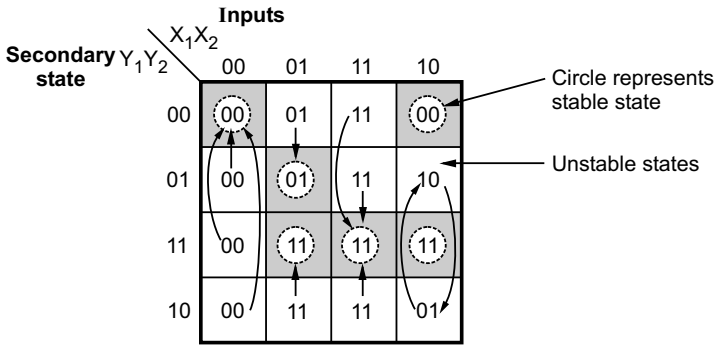
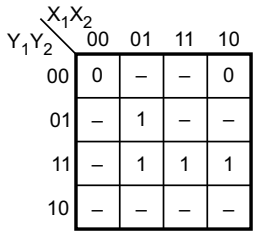


Fig. 9.5.5 (b) Transition table



Output is mapped for all stable states. For unstable states output is mapped unspecified

Fig. 9.5.5 (c) Output map

Example 9.5.4 *Derive the flow table for the circuit given in the Fig. 9.5.6.*

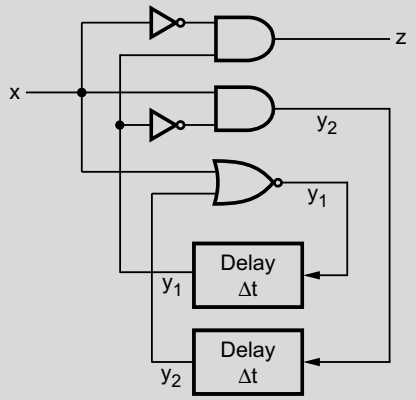


Fig. 9.5.6

Solution :

Step 1 : The excitation and output equations for the given circuit are :

$$Y_1 = \bar{x}\,\bar{y}_2$$

$$Y_2 = x\,\bar{y}_1$$

$$z = \bar{x}\,y_1$$

Present total state			Next total state			Stable total state	Z
S state		Input	NS state		Input		
Y_1	Y_2	X	Y_1	Y_2	X	Yes/No	
0	0	0	1	0	0	No	0
0	0	1	0	1	1	No	0
0	1	0	0	0	0	No	0
0	1	1	0	1	1	Yes	0
1	1	0	0	0	0	No	1
1	1	1	0	0	1	No	0
1	0	0	1	0	0	Yes	1
1	0	1	0	0	1	No	0

Table 9.5.4 State table

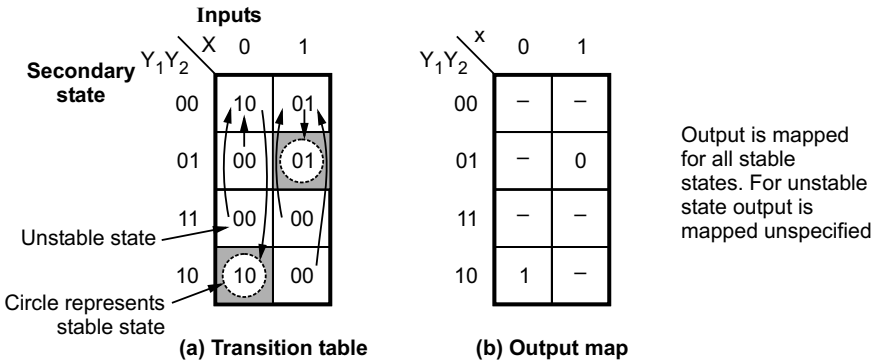


Fig. 9.5.7

Example for Practice

Example 9.5.5 *Analyse the fundamental mode asynchronous sequential circuit shown in Fig. 9.5.8.*

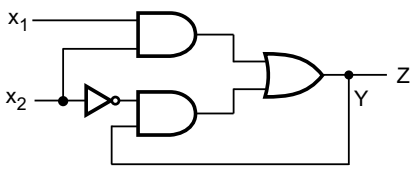


Fig. 9.5.8

Solution :

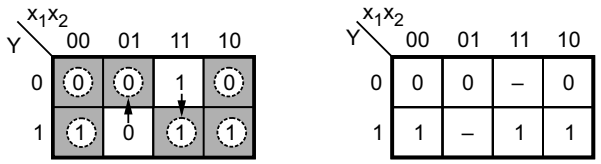


Fig. 9.5.9

Review Question

1. *List and explain the steps used for analyzing an asynchronous sequential circuit.*

Dec.-10, Marks 8

9.6 Design of Fundamental Mode Sequential Circuits

May-11, Dec.-08, 10, 11, 12

In the previous section we have seen the analysis of asynchronous sequential circuits. It gives us the information of how an existing sequential circuit works. The design process is an exactly reverse process. Here, we know the behaviour of the circuit and we have to develope the sequential circuit from scratch. Let us see the steps involved in designing of asynchronous sequential circuit.

1. Construction of a primitive flow table from the problem statement. An intermediate step may include the development of a state diagram.
2. Primitive flow table is reduced by eliminating redundant states by using state reduction techniques.
3. State assignment is made.
4. The primitive flow table is realized using appropriate logic elements.

9.6.1 Derivation of Primitive Flow Table

The flow table in the asynchronous sequential circuit is same as that of state table in the synchronous sequential circuit. In asynchronous sequential circuits state table is known as flow table because of the behaviour of the asynchronous sequential circuit. The state changes occur independent of a clock, based on the logic propagation delay, and cause the states to "flow" from one to another.

A primitive flow table is a special case of flow table. It is defined as a flow table which has exactly one stable state for each row in the table. The design process begins with the construction of primitive flow table. Let us see the following example to understand the process of construction of the primitive flow table from the problem statement.

Example 9.6.1 *Develop the state diagram and primitive row flow table for a logic system that has two inputs S and R and a single output Q. The device is to be an edge triggered SR flip-flop but without a clock. The device changes state on the rising edges of the two inputs. Static input values are not to have any effect in changing the Q output.*

Dec.-06, Marks 16

Solution : For SR flip-flop, initial state A is stable when no input changes have been detected. When SR input changes from 00 to 01 (Reset), the state transition occurs from A to C and when SR input changes from 00 to 10 (Set), the state transition occurs from A to B. This is illustrated in Fig. 9.6.1.

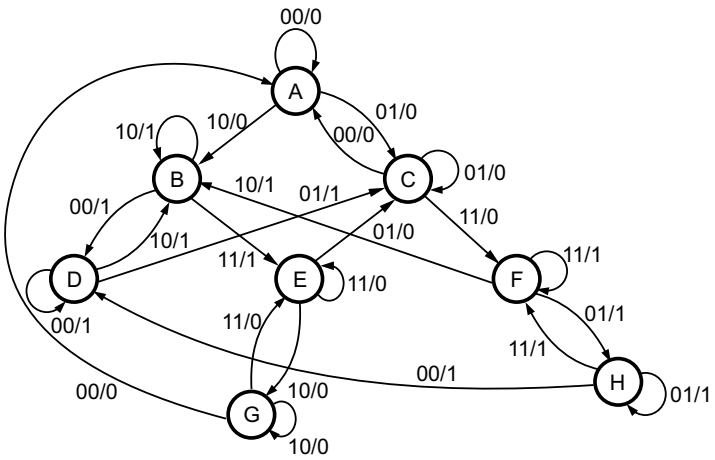


Fig. 9.6.1 State diagram

The state B is stable with a 10 static SR input and Q^+ is a 1. The state C is stable with a 01 static SR input and Q^+ remains a 0. A state transition from B to D occurs when the SR input changes from 10 to 00 with Q^+ output remains a 1. State change from B to E occurs when SR changes from 10 to 11. Once state E is reached the Q^+ output changes from a 1 to a 0. State change from D to C occurs when SR changes from 00 to 01. Once state C is reached the Q^+ output changes from a 1 to a 0. An SR input sequence of $00 \rightarrow 01 \rightarrow 11$ causes an $A \rightarrow C \rightarrow F$ state transition. The Q^+ remains a 0 unit the $01 \rightarrow 11$ SR change occurs. The 11 static input is allowed because only input rising edges cause Q changes. State transition F to B occurs when SR changes from 11 to 00.

The state change occurs from B to E when SR changes from $10 \rightarrow 11$, and the Q^+ output is a 1 for the transition. A state transition from state E to state G occurs when SR changes from 11 to 10 and Q^+ is 0. When SR input changes from $10 \rightarrow 11$ transition from G to E occurs. Once in state G, a 00 input on SR returns the state machine back to state A.

A state transition from state F to H occurs when SR changes from 11 to 01. For SR input 10 state transition F to B occurs.

Present state	Next state, for inputs SR/output Q			
	00	01	11	10
A	(A)/0	C/0	–	B/0
B	D/1	–	E/1	(B)/1
C	A/0	(C)/0	F/0	–
D	(D)/1	C/1	–	B/1
E	–	C/0	(E)/0	G/0
F	–	H/1	(F)/1	B/1
G	A/0	–	E/0	(G)/0
H	D/1	(H)/1	F/1	–

Table 9.6.1 (a) Primitive flow table

Table 9.6.1 (a) shows the primitive flow table constructed from the state diagram.

9.6.2 Reduction of Primitive Flow Table

The next step in the design process is to reduce the primitive flow table using state reduction techniques. Here, we are going to use merger graph technique to reduce primitive flow table.

Merger graphs is state reducing tool used to reduce states in the incompletely specified machine. The merger graph is defined as follows :

1. It contains the same number of vertices as the state table contains states. Refer Fig. 9.6.2.
2. Each compatible state pair is indicated by a line drawn between the two state vertices.
3. Every potentially compatible state pair, with outputs not in conflict but whose next states are different, is connected by a broken line. The implied states are drawn in the line break between the two potentially compatible states.
4. If two states are incompatible, no connecting line is drawn.

Example 9.6.2 *Reduce the primitive flow table shown in Table 9.6.2 using merger graph method.*

Present state	Next state , XY				Output Z
	00	01	11	10	
A	(A)	B	–	C	0
B	A	(B)	D	–	0
C	A	–	D	(C)	0
D	–	B	(D)	E	0
E	A	–	D	(E)	1

Table 9.6.2 Primitive flow table

Solution :

1. States A and B are compatible. Thus the line is drawn between A and B.
2. State A and C are compatible. Thus the line is drawn between A and C.
3. State B and C are compatible. Thus, the line is drawn between B and C.
4. States A and D are compatible only if implied states C and E are compatible. This is indicated by drawing a broken line between A and D with CE written in between.

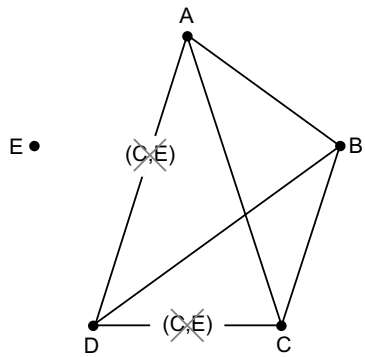


Fig. 9.6.2

5. States A and E are incompatible since there outputs are different, so line is not drawn between A and E. For the same reason states B, C and D are also not compatible with E.
6. State B and D are compatible. Thus, the line is drawn between B and D.
7. States C and D are compatible only if implied states C and E are compatible. This is indicated by drawing a broken line between C and D with CE written in between.
8. It is found that states C and E are not compatible and hence states A and D, and states D and C are also not compatible. This is indicated by cross (×) marks.

The compatibility lines of merger graph form a geometrical pattern consisting of one triangle (A, B, C), a line (B, D) and a single state E that is not compatible with any others. Thus maximum compatibles are :

(A, B, C) (B, D) (E)

Here, we can notice that state B is common in two sets. However, it can be compatible with either states A and C or state D, but not both. If we consider the next don't care state of B as state C, it is compatible with stable A and C. If we consider the next don't care state of B as state E it is compatible with state D.

Considering state B compatible with states A and C, we have following set of maximum compatibilities for given primitive flow table

(A, B, C) (D) (E)

Considering state B compatible with state D, we have following set of maximum compatibilities for given primitive flow table.

(A, C) (B, D) (E)

So we can say that there may be more than one possible way of merging rows when reducing a primitive flow table

According to alternative-I we have

$(A, B, C) \rightarrow S_0$

$(D) \rightarrow S_1$

$(E) \rightarrow S_2$

and according to alternative-II we have

$(A, C) \rightarrow S_0$

$(B, D) \rightarrow S_1$

$(E) \rightarrow S_2$

The Table 9.6.2 shows the reduced primitive flow table using both the alternatives.

Present state	Next state , XY				Output Z
	00	01	11	10	
S_0	(S0)	(S0)	S_1	(S0)	0
S_1	–	S_0	(S1)	S_2	0
S_2	S_0	–	S_1	(S2)	1

Reduced primitive flow table (Alternative I)

Present state	Next state , XY				Output Z
	00	01	11	10	
S_0	(S0)	S_1	S_1	(S0)	0
S_1	S_0	(S1)	(S1)	S_2	0
S_2	S_0	–	S_1	(S2)	1

Reduced primitive flow table (Alternative II)

Table 9.6.2 (a)

Example 9.6.3 *Reduce the primitive flow table derived in example 9.6.1.*

Solution : The derived primitive flow table is redrawn in Table 9.6.3.

The Fig. 9.6.3 shows the merger graph. Each compatible state pair is indicated by a line drawn between the two states vector.

Every potentially compatible state pair, with outputs not in conflict but whose next states are different, is connected by a broken line. The implied states are drawn in the line break between the two potentially compatible states. If two states are incompatible, no connecting line is drawn.

Present state	Next state, for inputs SR/output Q			
	00	01	11	10
A	(A)/0	C/0	–	B/0
B	D/1	–	E/1	(B)/1
C	A/0	(C)/0	F/0	–
D	(D)/1	C/1	–	B/1
E	–	C/0	(E)/0	G/0
F	–	H/1	(F)/1	B/1
G	A/0	–	E/0	(G)/0
H	D/1	(H)/1	F/1	–

Table 9.6.3

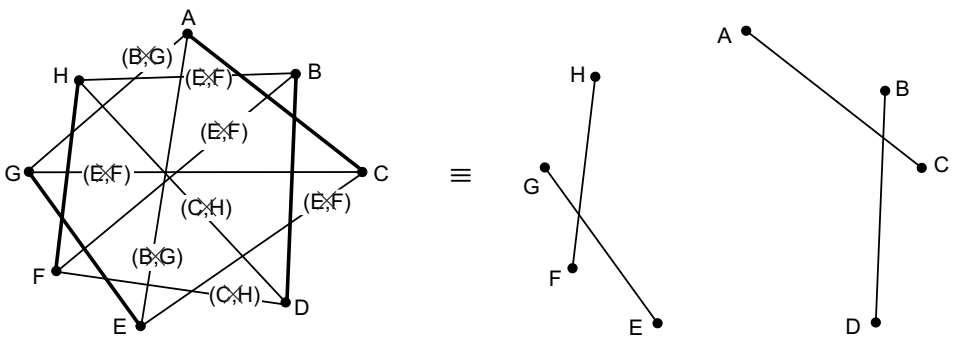


Fig. 9.6.3 Merger graph for primitive flow table

Present state	Next State, for inputs SR/output Q			
	00	01	11	10
A	(A)/0	C/0	**F/0**	B/0
B	D/1	**C/1**	E/1	(B)/1
C	A/0	(C)/0	F/0	**B/0**
D	(D)/1	C/1	**E/1**	B/1
E	**A/0**	C/0	(E)/0	G/0
F	**D/1**	H/1	(F)/1	B/1
G	A/0	**C/0**	E/0	(G)/0
H	D/1	(H)/1	F/1	**B/1**

Equivalent states (S_1): B, D

Equivalent states (S_0): A, C

Equivalent states (S_2): E, G

Equivalent states (S_3): F, H

Table 9.6.4

Therefore, we have

(A, C) $\rightarrow S_0$

(B, D) $\rightarrow S_1$

(E, G) $\rightarrow S_2$

(F, H) $\rightarrow S_3$

The Table 9.6.5 shows the reduced primitive flow.

Present state	Next state for inputs SR/output Q			
	00	01	11	10
S_0	(S_0), 0	(S_0), 0	S_3, 0	S_1, 0
S_1	(S_1), 1	S_0, 1	S_2, 1	(S_1), 1
S_2	S_0, 0	S_0, 0	(S_2), 0	(S_2), 0
S_3	S_1, 1	(S_3), 1	(S_3), 1	S_1, 1

Table 9.6.5 Reduced primitive flow Table

The Fig. 9.6.4 shows the reduced state diagram for primitive flow table.

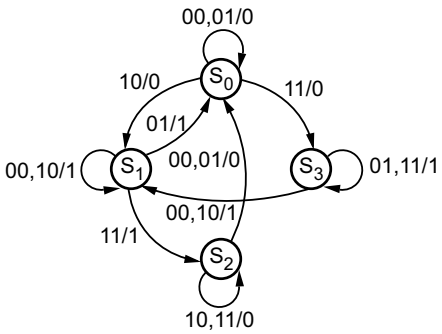


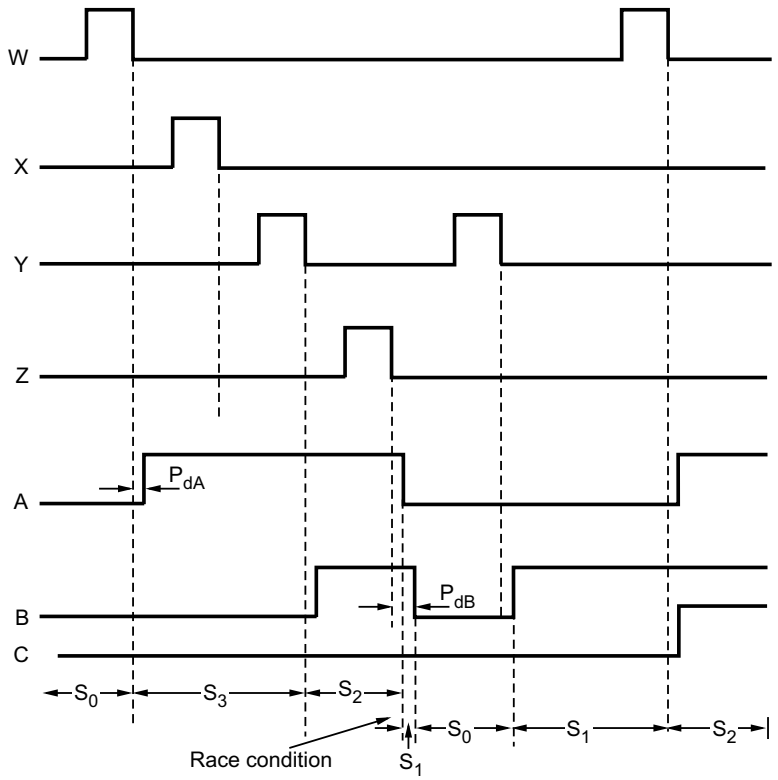
Fig. 9.6.4

9.6.3 Race Free State Assignment

The state assignment step in asynchronous circuits is essentially the same as it is for synchronous circuits, except for one difference. In synchronous circuits, the state assignments are made with the objective of circuit reduction. In asynchronous circuits, the objective of state assignment is to avoid critical races.

9.6.3.1 Races and Cycles

When two or more binary state variables change their value in response to a change in an input variable, race condition occurs in an asynchronous sequential circuit. In case of unequal delays, a race condition may cause the state variables to change in an unpredictable manner. For example, if there is a change in two state variables due to change in input variable such that both change from 00 to 11. In this situation, the difference in delays may cause the first variable to change faster than the second resulting the state variables to change in sequence from 00 to 10 and then to 11. On the other hand, if the second variable changes faster than the first, the state variables change from 00 to 01 and then to 11. If the final stable state that the circuit reaches does not depend on the order in which the state variable changes, the race condition is not harmful and it is called a **noncritical race**. But, if the final stable state depends on the order in which the state variable changes, the race condition is harmful and it is called a **critical race**. Such critical races must be avoided for proper operation. Let us see the examples of noncritical races and critical races.



Note: P_{dA} : Propagation delay for A
P_{dB} : Propagation delay for B

Fig. 9.6.5 Timing diagram

Noncritical Races

Fig. 9.6.6 illustrates noncritical races. It shows transition tables in which X is a input variable and y_1 y_2 are the state variables. Consider a circuit is in a stable state $y_1y_2x = 000$ and there is a change in input from 0 to 1. With this change in the input there are three possibilities that the state variables may change. They can either change simultaneously from 00 to 11, or they may change in sequence from 00 to 01 and then to 11, or they may change in sequence from 00 to 10 and then to 11. In all cases, the final stable state is 11, which results in a noncritical race condition. In Fig. 9.6.6 (b) final stable state is y_1y_2x = 101.

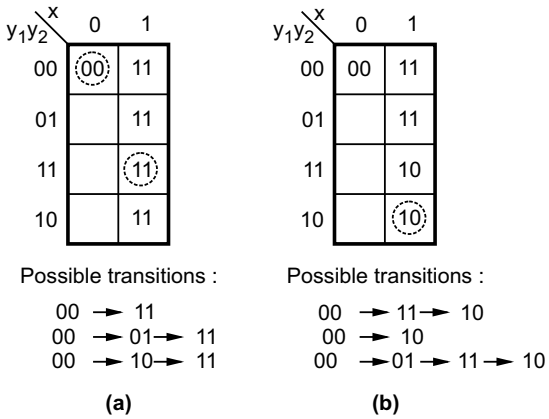


Fig. 9.6.6 Examples of noncritical races

Critical Races

Fig. 9.6.7 illustrates critical race. Consider a circuit is in a stable state y_1y_2x = 000 and there is a change in input from 0 to 1. If state variables change simultaneously, the final stable state is y_1y_2x = 111. If Y_2 changes to 1 before Y_1 because of unequal propagation delay, then the circuit goes to the stable state 011 and remain there. On the other hand, if Y_1 changes faster than Y_2, then the circuit goes to the stable state 101 and remain there. Hence, the race is critical because the circuit goes to different stable states depending on the order in which the state variables change.

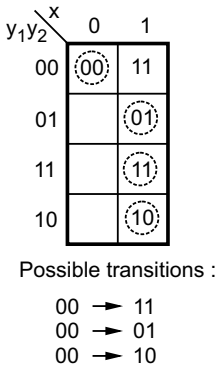


Fig. 9.6.7 Example of critical race

Cycles

A cycle occurs when an asynchronous circuit makes a transition through a series of unstable states. When a state assignment is made so that it introduces cycles, care must be taken to ensure that each cycle terminates on a stable state. If a cycle does not contain a stable state, the circuit will go from one unstable state to another, until the inputs are changed. Obviously, such a situation must always be avoided when designing asynchronous circuits.

Two techniques are commonly used for making a critical race free state assignment.

1. Shared row state assignment.
2. One hot state assignment.

9.6.3.2 Shared Row State Assignment

Races can be avoided by making a proper binary assignment to the state variables. Here, the state variables are assigned with binary numbers in such a way that only one state variable can change at any one time when a state transition occurs. To accomplish this, it is necessary that states between which transition occur be given adjacent assignments. Two binary values are said to be adjacent if they differ in only one variable. For example, 110 and 111 are adjacent because they differ only in the third bit.

Fig. 9.6.8 shows the transition diagram. The transition diagram shows that there is transition from state a to state b and transition from state a to state c. The state a is assigned binary value 00 and state c is assigned binary value 11. This assignment will cause a critical race during the transition from a to c because there are two changes in the binary state variables. A race free assignment can be obtained by introducing addition binary state say d with binary value 10, which is adjacent to both a and c. Fig. 9.6.9 shows the modified transition diagram. As shown in the Fig. 9.6.9, the transition from a to c will go through d. This causes the binary variables to change from 00 $\rightarrow$ 10 $\rightarrow$ 11 which satisfy the condition that only one binary variable changes during each state transition, thus avoiding the critical race.

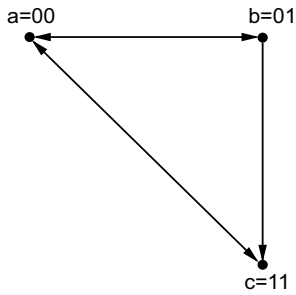


Fig. 9.6.8 Transition diagram

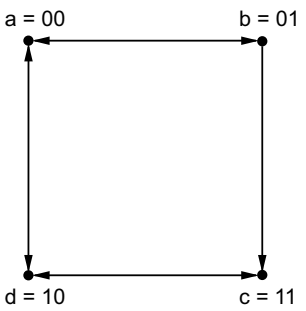


Fig. 9.6.9 Transition diagram with race free state assignment

This technique is called shared row state assignment because in this technique extra state, i.e. extra row is introduced in a flow table. This extra state is shared between two stable states.

9.6.3.3 One Hot State Assignment

The one hot state assignment is an another method for finding a race free state assignment. In this method, only one variable is active or 'hot' for each row in the original flow table, i.e. it requires one state variable for each row of the flow table. Additional rows are introduced to provide single variable changes between internal state transitions. This is illustrated in the following example.

State variables				State	Inputs X_1 X_2			
F_4	F_3	F_2	F_1		00	01	11	10
0	0	0	1	A	(A)	B	C	C
0	0	1	0	B	A	(B)	C	D
0	1	0	0	C	A	B	(C)	(C)
1	0	0	0	D	(D)	B	C	(D)

Fig. 9.6.10 Flow table

Consider a flow table given in Fig. 9.6.10 four state variables are used to represent the four rows in the table. Each

row is represented by a case where only one of the four state variables is a 1. A transition from state A to state B requires two state variable changes; F_1 from 1 to 0 and F_2 from 0 to 1. By directing the transition A to B through a new row E which contains 1s where both states A and B have 1s. We require only one state variable change from transition A to E and then from transition E to B. This permits the race free transition between A and B.

In general, we can say that, in row i of the table, state variable F_i is 1 and all other state variables are 0. When a transition between row i and row j is required, first state variable F_j is set to 1 (so that both F_i and F_j are 1), and then F_i is set to 0. Thus each transition between two rows in the flow table goes through one intermediate row. This permits the race free transition but requires two state transition times.

The Fig. 9.6.11 shows the complete one hot state assignment flow table. When $X_1X_2 = 01$ the transition from A to B is passing through the dummy state E. Similarly, when $X_1X_2 = 00$ the transition from C to A is passing through the dummy state F and so on. The original table thus gets modified and it is as shown in Fig. 9.6.11.

	State variables				State	Inputs X_1 X_2			
	F_4	F_3	F_2	F_1		00	01	11	10
Original table	0	0	0	1	A	(A)	~~B~~E	~~C~~F	~~C~~F
	0	0	1	0	B	~~A~~E	(B)	C	D
	0	1	0	0	C	~~A~~F	~~B~~G	(C)	(C)
	1	0	0	0	D	(D)	~~B~~H	~~C~~I	(D)
Added rows	0	0	1	1	E	A	B	–	–
	0	1	0	1	F	A	–	C	C
	0	1	1	0	G	–	B	C	–
	1	0	1	0	H	–	B	–	D
	1	1	0	0	I	–	–	C	–

Fig. 9.6.11 One hot state assignment flow table

9.6.4 Realization of Flow Table

To understand the process of realization of flow table we see the following example. The example illustrates all the steps of designing of asynchronous sequential circuit.

Examples for Understanding

Example 9.6.4 *Design an asynchronous sequential circuit with two inputs X and Y and with one output Z. Whenever Y is 1, input X is transferred to Z. When Y is 0, the output does not change for any change in X.* **May-03, Marks 16**

Solution :

Step 1 : Draw state diagram and derive primitive flow table.

The state diagram for above problem statement can be given as shown in Fig. 9.6.12.

A primitive flow table is constructed from the state diagram shown in Fig. 9.6.13.

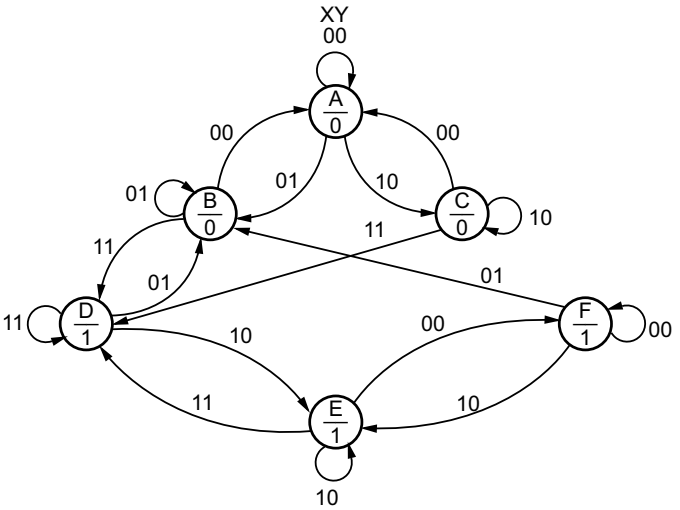
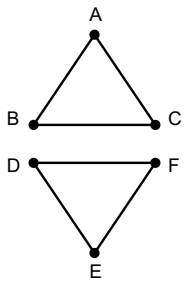


Fig. 9.6.12 State diagram for example problem

Present state	Next state for X Y Inputs				Output z
	00	01	11	10	
A	(A)	B	–	C	0
B	A	(B)	D	–	0
C	A	–	D	(C)	0
D	–	B	(D)	E	1
E	F	–	D	(E)	1
F	(F)	B	–	E	1

Fig. 9.6.13 Primitive flow table for example problem

Step 2 : Reduction of primitive flow table.



(a) Merger graph for example problem

Present state	Next state for X Y Inputs				Output z	
	00	01	11	10		
A	(A)	B	**D**	C	0	Equivalent states (S_0)
B	A	(B)	D	**C**	0	
C	A	**B**	D	(C)	0	
D	**F**	B	(D)	E	1	Equivalent states (S_1)
E	F	**B**	D	(E)	1	
F	(F)	B	**D**	E	1	

(b) Assignment of don't care states

Fig. 9.6.14

Above primitive flow table can be reduced using merger graph as shown in Fig. 9.6.14. Here, six vertices are drawn corresponding to six states, and complete line between states vertices is drawn for compatible states. The merger graph shown in Fig. 9.6.14 gives the two compatible pairs as a set of maximal compatibles

$(A, B, C) \rightarrow S_0$

$(D, E, F) \rightarrow S_1$

This set of maximal compatibles covers all of the original states resulting in reduced flow table as shown in Fig. 9.6.15.

Present state	Next state, for XY Inputs				Output Z
	00	01	11	10	
S_0	(S_0)	(S_0)	S_1	(S_0)	0
S_1	(S_1)	S_0	(S_1)	(S_1)	1

Fig. 9.6.15 Reduced flow table for example problem

Step 3 : State assignment

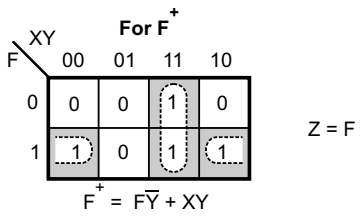
In order to obtain the circuit described by the reduced flow table, it is necessary to assign a distinct binary value to each state. This assignment converts the flow table into a transition table. This state assignment should ensure that the circuit will be free of critical races. In this reduced flow table we have only two rows and fortunately, there cannot be critical races when row in the flow table are two. Therefore, we can assign 0 to state S_0 and 1 to state S_1 to get transition table as shown in Fig. 9.6.16.

Present state F	Next state (F+), for XY Inputs				Output z
	00	01	11	10	
0	(0)	(0)	1	(0)	0
1	(1)	0	(1)	(1)	1

Fig. 9.6.16 Transition table for example problem

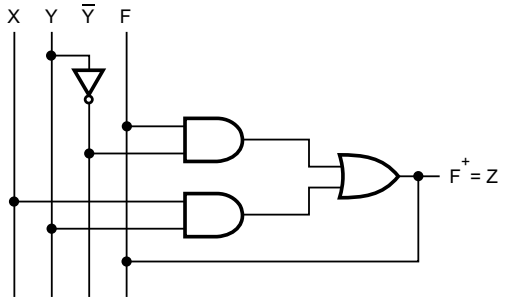
Step 4 : Realization of circuit using logic elements

The Boolean expressions for function and the output are derived using K-map simplification. Then each boolean expression is implemented using logic gates, as shown below.



(a) K-map simplification

Z = F



(b) Logic diagram

Fig. 9.6.17

Step 5 : Realization of circuit using SR latch.

We can also implement the circuit represented by transition table using SR latch. In this case we have to derive the input expressions for S and R inputs of SR latch. To derive the input expressions first we have to obtain the K-map for S and R by referring the excitation table of SR latch and then solve the K-map for S and R, individually. This is illustrated in Fig. 9.6.18. Here, we have to see the transition from transition table and obtain the SR inputs for the transition by referring excitation table of SR latch. For example, for input XY = 01, the second row of transition table shown in Fig. 9.6.18 requires a transition from F = 1 to F^+ = 0. The excitation table specifies S = 0, R = 1 for this change. Therefore, the corresponding square in the S map is marked with a 0 and the one in the R map with a 1. All other squares are filled with values in a similar manner.

F	F^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Fig. 9.6.18 (a) Excitation table for SR latch

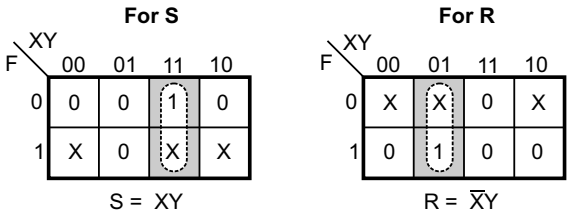


Fig. 9.6.18 (b) K-maps for S and R

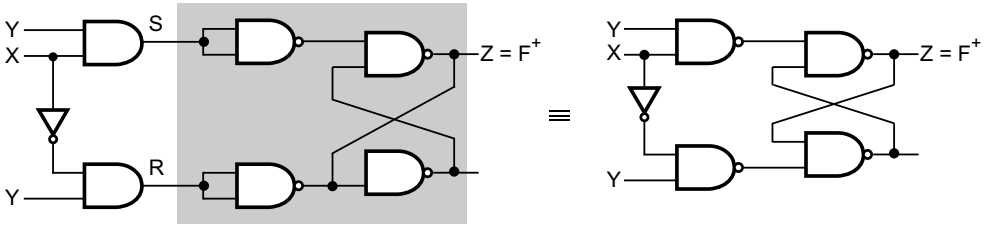


Fig. 9.6.19

Example 9.6.5 *Design an asynchronous sequential circuit that has two inputs X_2 and X_1 and one output Z. When X_1 = 0, the output Z is 0. The first change in X_2 that occurs while X_1 is 1 will cause output Z to be 1. The output Z will remain 1 until X_1 returns to 0.*

Dec.-08, May-09, Marks 16

Solution :

Step 1 : Draw state diagram and derive primitive flow table.

The state diagram for above problem statement can be given as shown in Fig. 9.6.20.

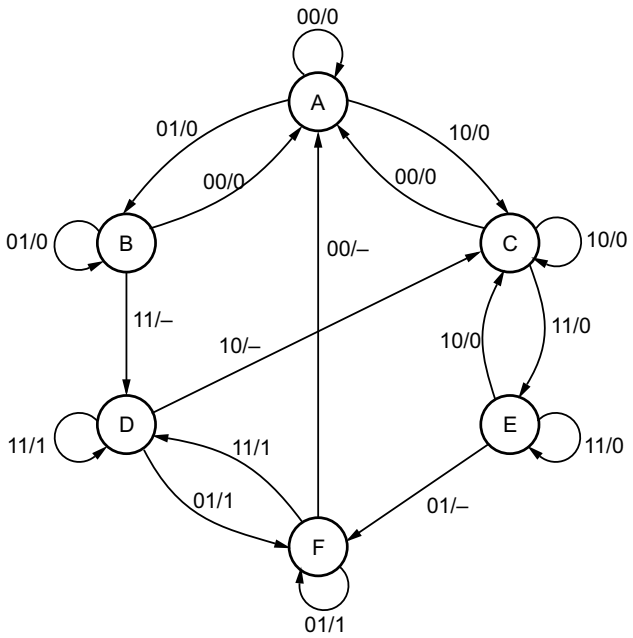


Fig. 9.6.20 State diagram for given problem

A primitive flow table is constructed from the state diagram shown in Fig. 9.6.21. The merger graph for above primitive flow table can be given as in Fig. 9.6.22.

Present state	Next state, Output Z for X_2X_1 Inputs			
	00	01	11	10
A	(A), 0	B , 0	– , –	C , 0
B	A , 0	(B), 0	D , –	– , –
C	A , 0	– , –	E , 0	(C), 0
D	– , –	F , 1	(D), 1	C , –
E	– , –	F , –	(E), 0	C , 0
F	A , –	(F), 1	D , 1	– , –

Fig. 9.6.21 Primitive flow table for given problem

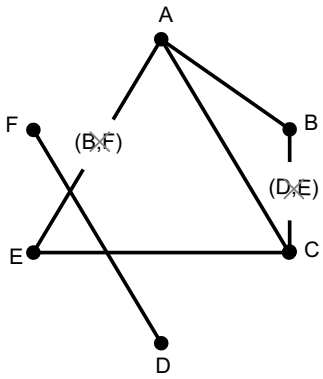


Fig. 9.6.22 Merger graph

Present state	Next state, Output Z for X_2X_1 Inputs			
	00	01	11	10
A	(A), 0	B , 0	**D , –**	C , 0
B	A , 0	(B), 0	D , –	**C , –**
C	A , 0	**F , –**	E , 0	(C), 0
D	**A , –**	F , 1	(D), 1	C , –
E	**A , 0**	F , –	(E), 0	C , 0
F	A , –	(F), 1	D , 1	**C , –**

Equivalent state (S_0): A, B
Equivalent state (S_1): C, E
Equivalent state (S_2): D, F

Fig. 9.6.23 Assignments of don't care states

Step 2 : Reduction of primitive flow table

The merger graph gives the two compatible pairs as a set of maximal compatibles.

$(A, B) \rightarrow S_0$

$(C, E) \rightarrow S_1$

$(D, F) \rightarrow S_2$

This set of maximal compatible covers all of the original states resulting in the reduced flow table as shown in Fig. 9.6.24.

Present state	Next state, Output Z for X_2X_1 Inputs			
	00	01	11	10
S_0	(S_0), 0	(S_0) , 0	S_2 , –	S_1 , 0
S_1	S_0 , 0	S_2 , –	(S_1), 0	(S_1), 0
S_2	S_0 , –	(S_2) , 1	(S_2), 1	S_1 , –

Fig. 9.6.24 Reduced flow table

Step 3 : State assignment

Now if we assign $S_0 \rightarrow 00$, $S_1 \rightarrow 01$ and $S_2 \rightarrow 10$ then we need one more state $S_3 \rightarrow 11$ to prevent critical race during transition of $S_1 \rightarrow S_2$ or $S_2 \rightarrow S_1$. By introducing S_3 the transitions $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_1$ are routed through S_3. Thus, after state assignment the flow table can be given as shown in Fig. 9.6.25.

Present state F_2 F_1	Next state, Output Z for X_2X_1 Inputs			
	00	01	11	10
$S_0 \rightarrow$ 0 0	(S_0), 0	(S_0) , 0	S_2 , –	S_1 , 0
$S_1 \rightarrow$ 0 1	S_0 , 0	S_3 , –	(S_1), 0	(S_1) , 0
$S_2 \rightarrow$ 1 0	S_0 , –	(S_2) , 1	(S_2), 1	S_3 , –
$S_3 \rightarrow$ 1 1	– , –	S_2 , –	– , –	S_1 , –

Fig. 9.6.25 Flow table with state assignment

The flow table in Fig. 9.6.25 can be converted to a transition table as shown in Fig. 9.6.26.

Present state		Next state, Output Z for X_2X_1 Inputs			
F_2	F_1	00	01	11	10
0	0	00 , 0	00 , 0	10 , –	01 , 0
0	1	00 , 0	11 , –	01 , 0	01 , 0
1	0	00 , –	10 , 1	10 , 1	11 , –
1	1	– , –	10 , –	– , –	01 , –

Fig. 9.6.26 Transition table

Step 4 : Realization of circuit using logic elements

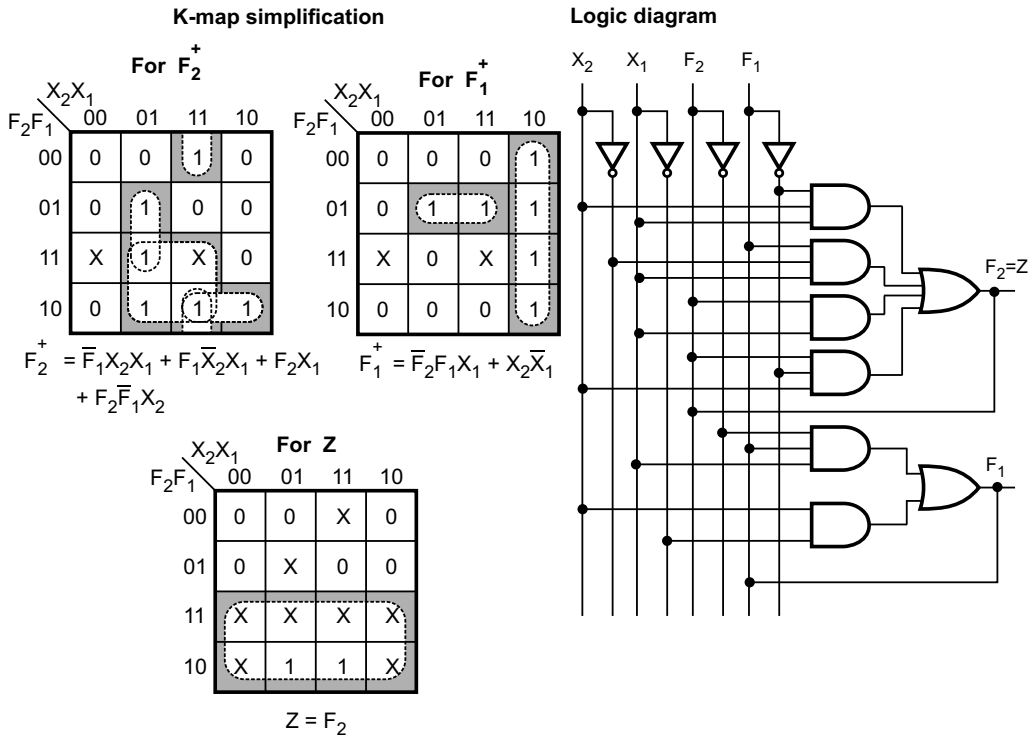


Fig. 9.6.27

Example 9.6.6 *Design a two-input (x_1, x_2), two-output (z_1, z_2) fundamental-mode circuit that has the following specifications. When $x_1x_2 = 00$, $z_1z_2 = 00$. The output 10 will be produced following the occurrence of the input sequence 00-01-11. The output will remain at 10 until the input returns to 00 at which time it becomes 00. An output of 01 will be produced following the receipt of the input sequence 00-10-11. And once again, the output will remain at 01 until a 00 input occurs, which returns the output to 00.*

May-08, Marks 16

Solution :

Step 1 : Draw state diagram and derive primitive flow table.

The state diagram for the given problem is as shown in the Fig. 9.6.28.

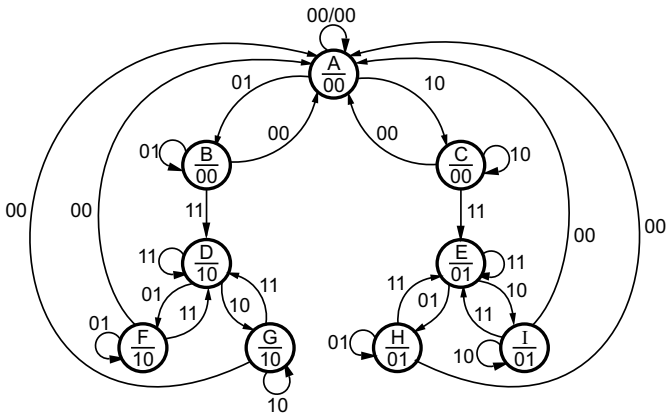


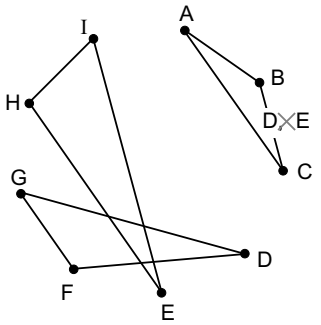
Fig. 9.6.28

The Table 9.6.6 shows the primitive flow table constructed from the state diagram.

Present state	Next state X_1X_2				Outputs Z_1 Z_2
	00	01	11	10	
A	(A)	B	–	C	00
B	A	(B)	D	–	00
C	A	–	E	(C)	00
D	–	F	(D)	G	10
E	–	H	(E)	I	01
F	A	(F)	D	–	10
G	A	–	D	(G)	10
H	A	(H)	E	–	01
I	A	–	E	(I)	01

Table 9.6.6 Primitive flow table

Step 2 : Reduce primitive flow table.



(a) Merger graph

Present state	Next state X_1X_2 00	01	11	10	Outputs Z_1 Z_2
A	(A)	B	D	C	00
B	A	(B)	D	C	00
C	A	–	E	(C)	00
D	A	F	(D)	G	10
E	A	H	(E)	I	01
F	A	(F)	D	G	10
G	A	F	D	(G)	10
H	A	(H)	E	I	01
I	A	H	E	(I)	01

Equivalent states (S_0)

(S_1)

Equivalent states (S_2)

Equivalent states (S_3)

(b) Assignment of don't care states

Fig. 9.6.29

The merger graph gives four compatible pair as a set of maximum compatibilities.

(A, B) $\rightarrow S_0$

(C) $\rightarrow S_1$

(D, F, G) $\rightarrow S_2$

(E, H, I) $\rightarrow S_3$

The Table 9.6.7 shows the reduced primitive flow table.

Present state	Next state X_1X_2 00	01	11	10	Outputs Z_1 Z_2
S_0	(S_0)	(S_0)	S_2	S_1	00
S_1	S_0	–	S_3	(S_1)	00
S_2	S_0	(S_2)	(S_2)	(S_2)	10
S_3	S_0	(S_3)	(S_3)	(S_3)	01

Table 9.6.7 Reduced primitive flow table

Step 3 : State assignment

We assign : S_0 = 00, S_1 = 01, S_2 = 10 and S_3 = 11

The Table 9.6.8 shows the transition table with state assignment.

Present State		Next state X_1 X_2						Output	
F_1	F_2	0 0	0 1	1 1	1 0	Z_1	Z_2
0	0	0 0	0 0	1 0	0 1	0	0
0	1	0 0	–	1 1	0 1	0	0
1	0	0 0	1 0	1 0	1 0	1	0
1	1	0 0	1 1	1 1	1 1	0	1

Table 9.6.8 Transition table

Step 4 : Realization of circuit using logic elements.

K-map simplification

F_1F_2 \ X_1X_2	00	01	11	10
00	0	0	1	0
01	0	X	1	0
11	0	1	1	1
10	0	1	1	1

$$F_1^+ = X_1X_2 + F_1X_2 + F_1X_1$$

F_1F_2 \ X_1X_2	00	01	11	10
00	0	0	0	1
01	0	X	1	1
11	0	1	1	1
10	0	0	0	0

$$F_2^+ = \overline{F}_1X_1\overline{X}_2 + F_2X_2 + F_2X_1$$

$$Z_1 = F_1\,\overline{F}_2 \qquad Z_2 = F_1\,F_2$$

Fig. 9.6.30

Logic diagram

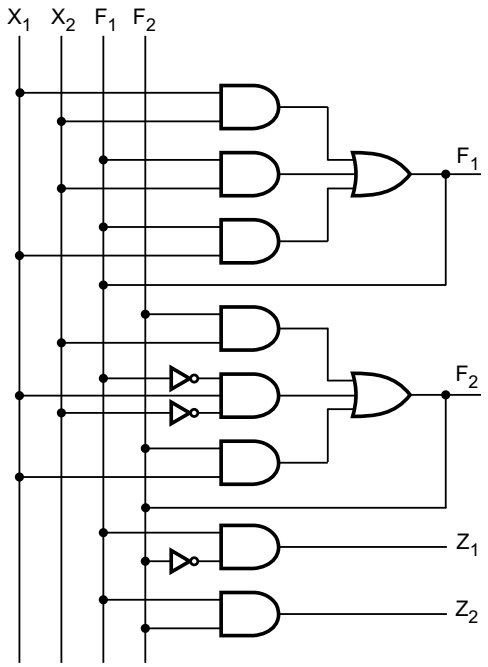


Fig. 9.6.31

Examples with Solutions

Example 9.6.7 *Obtain a primitive flow table for a circuit with two inputs x_1 and x_2 and two outputs z_1 and z_2 that satisfies the following four conditions.*

i) When $x_1x_2 = 00$ output $z_1z_2 = 00$.

ii) When $x_1 = 1$ and x_2 changes from 0 to 1, the output $z_1z_2 = 01$.

iii) When $x_2 = 1$ and x_1 changes from 0 to 1, the output $z_1z_2 = 10$.

iv) Otherwise the output does not change.

Solution : The state diagram for above problem statement can be given as shown in Fig. 9.6.32.

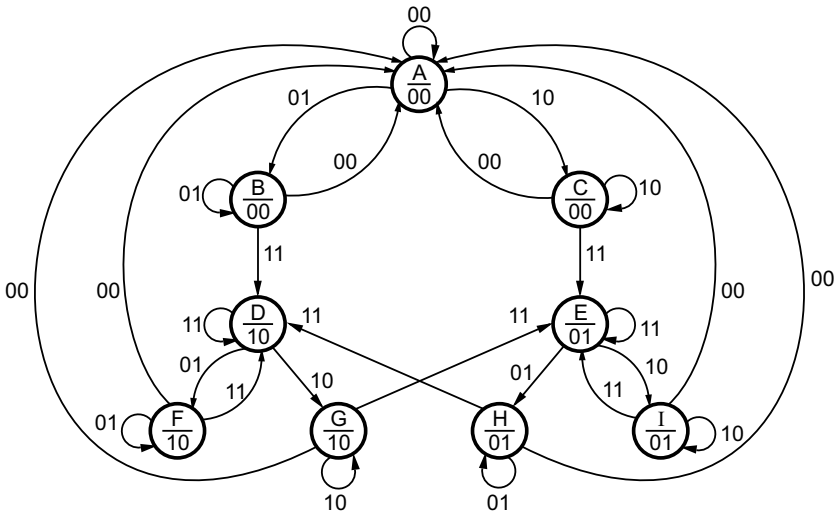


Fig. 9.6.32

A primitive flow table is constructed from the state diagram shown in Fig. 9.6.33.

Present state	X_1, X_2 Inputs Next state				Outputs $Z_1 Z_2$
	00	01	11	10	
A	(A)	B	–	C	00
B	A	(B)	D	–	00
C	A	–	E	(C)	00
D	–	F	(D)	G	10
E	–	H	(E)	I	01
F	A	(F)	D	–	10
G	A	–	E	(G)	10
H	A	(H)	D	–	01
I	A	–	E	(I)	01

Fig. 9.6.33 Primitive flow table

Example 9.6.8 *Design a T flip-flop from logic gates.* **Dec.-07, 08, 09, Marks 16**

Solution : The T flip-flop has one excitation input and one clock input. But here we use another input P that will function as a clock. The flip-flop will change state if T = 1 and when the clock (P) changes from 1 to 0. Under all other input conditions, output Q will remain constant. We assume that T and P do not change simultaneously.

Step 1 : Draw state diagram and derive primitive flow table.

The state diagram for above problem is as shown in Fig. 9.6.34.

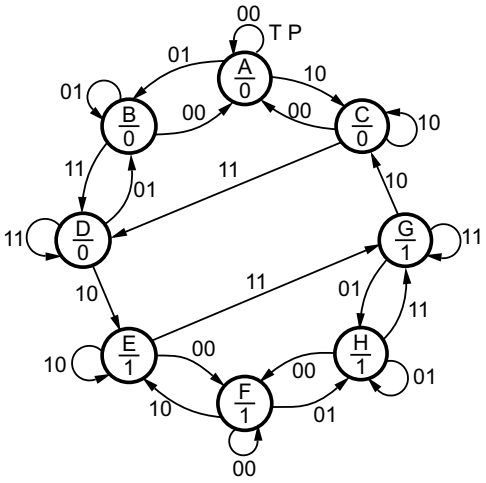
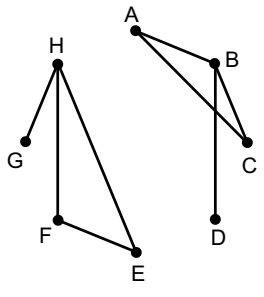


Fig. 9.6.34 State diagram

Present state	Next state, for T P inputs				Output Q
	00	01	11	10	
A	(A)	B	–	C	0
B	A	(B)	D	–	0
C	A	–	D	(C)	0
D	–	B	(D)	E	0
E	F	–	G	(E)	1
F	(F)	H	–	E	1
G	–	H	(G)	C	1
H	F	(H)	G	–	1

Fig. 9.6.35 Primitive flow table

Step 2 : Reduction of primitive flow table.



Merger graph

Present state	Next state, for T P inputs				Output Q	
	00	01	11	10		
A	(A)	B	**D**	C	0	Equivalent states (S_0)
B	A	(B)	D	**C**	0	
C	A	**B**	D	(C)	0	
D	–	B	(D)	E	0	← (S_1)
E	F	**H**	G	(E)	1	Equivalent states (S_2)
F	(F)	H	**G**	E	**1**	
G	–	H	(G)	C	1	← (S_3)
H	F	(H)	G	**E**	1	

Fig. 9.6.36 Assignments of don't care states

The merger graph shown in Fig. 9.6.36 gives the four compatible pairs as a set of maximal compatibles.

(A, B, C) → S_0 D → S_1

(E, F, H) → S_2 G → S_3

This set of maximum compatibles covers all of the original states resulting in the reduced flow table as shown in the Fig. 9.6.37.

Present state	Next state, Output Q for T P Inputs			
	00	01	11	10
S_0	(S_0), 0	(S_0), 0	S_1, –	(S_0), 0
S_1	–, –	S_0, –	(S_1), 0	S_2, –
S_2	(S_2), 1	(S_2), 1	S_3, –	(S_2), 1
S_3	–, –	S_2, –	(S_3), 1	S_0, –

Fig. 9.6.37 Reduced primitive flow table

Present state		Next state, for T P inputs				Output Q
F_2	F_1	00	01	11	10	
0	0	00	00	01	00	0
0	1	–	00	01	11	0
1	1	11	11	10	11	1
1	0	–	11	10	00	1

Table 9.6.9 Transition table

Step 3 : State assignment

By making state assignment as $S_0 \rightarrow 00$, $S_1 \rightarrow 01$, $S_2 \rightarrow 11$ and $S_3 \rightarrow 10$ we can avoid all the races. Substituting state assigned values to state we get transition table as shown in Fig. 9.6.9.

Step 4 : Realization of circuit using logic elements

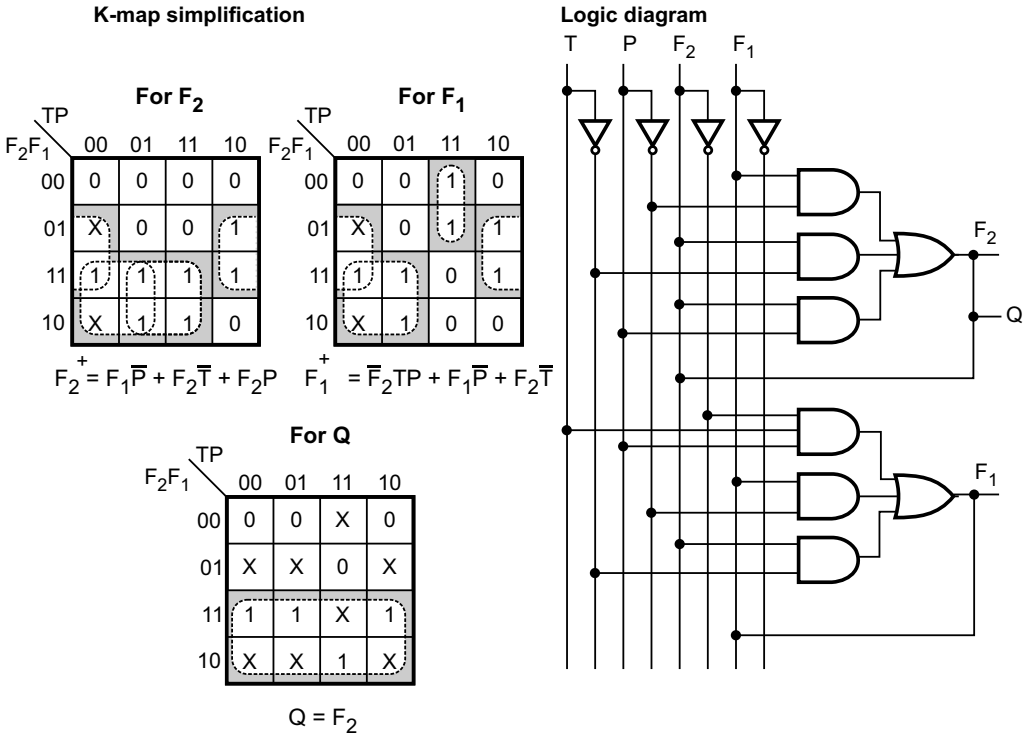


Fig. 9.6.38 Transition table

Example 9.6.9 *Design a circuit with inputs A and B to give an output Z = 1 when AB = 11 but only if A becomes 1 before B, by drawing total state diagram, primitive flow table and output map in which transient state is included.* **May-06**

Solution :

Step 1 : Draw state diagram and derive primitive flow table.

The state diagram for above problem statement can be given as shown in the Fig. 9.6.39.

A primitive flow table is constructed from the state diagram as shown in the Fig. 9.6.40.

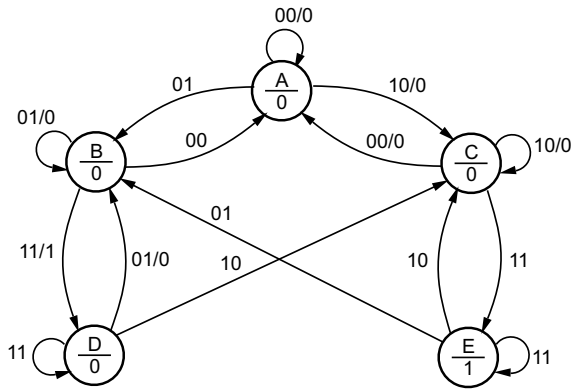
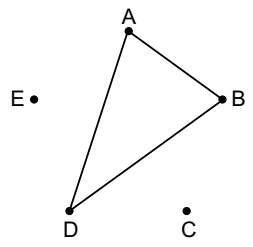


Fig. 9.6.39 Total state table

Present state	Next state for AB inputs				Output Z
	00	01	11	10	
A	(A)	B	–	C	0
B	A	(B)	D	–	0
C	A	–	E	(C)	0
D	–	B	(D)	C	0
E	–	B	(E)	C	1

Fig. 9.6.40 Primitive flow table for given problem

Step 2 : Reduction to primitive flow table.



(a) Merger graph

Present state	Next state , Output Z for AB inputs				Output Z
	00	01	11	10	
A	(A)	B	**D**	C	0
B	A	(B)	D	**C**	0
C	A	**B**	**E**	(C)	0
D	**A**	B	(D)	C	0
E	**A**	B	(E)	**C**	1

Equivalent state (S_0)

(b)

Fig. 9.6.41

The merger graph gives three compatible pairs as a set of maximum compatibilities.

$(A, B, D) \rightarrow S_0$

$C \rightarrow S_1$ $E \rightarrow S_2$

Present state	Next state for AB inputs				Output Z
	00	01	11	10	
S_0	(S_0)	(S_0)	(S_0)	S_1	0
S_1	S_0	S_0	S_2	(S_1)	0
S_2	S_0	S_0	(S_2)	S_1	1

Fig. 9.6.42 Reduced primitive flow table

Step 3 : State assignment

We assign $S_0 = 00$, $S_1 = 01$ and $S_2 = 10$

Present state F_1 F_2	Next state for AB input				Output Z
	00	01	11	10	
0 0	00	00	00	01	0
0 1	00	00	10	01	0
1 0	00	00	10	01	1

Fig. 9.6.43 Transition table

Step 4 : Realization of circuit using logic elements.

K-Map simplification

For F_1

F_1F_2 \ AB	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	0	0	1	0

$F_1^+ = F_1 AB + F_2 AB$

For F_2

F_1F_2 \ AB	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	X	X	X	X
10	0	0	0	1

$F_2^+ = A\overline{B}$

Logic diagram

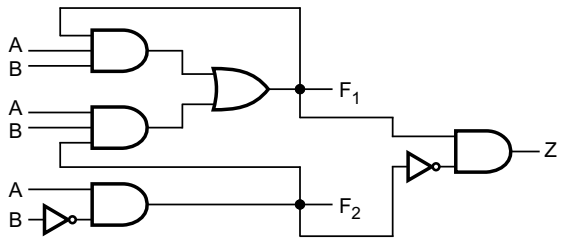


Fig. 9.6.44

Example 9.6.10 *Design a asynchronous D-type latch with two inputs G and D and output Q. Assume fundamental mode of operation.* **Dec.-11, Marks 16**

Solution : Since it is a fundamental mode operation, we assume that G and D do not change simultaneously.

Step 1 : Draw state diagram and derive primitive flow table.

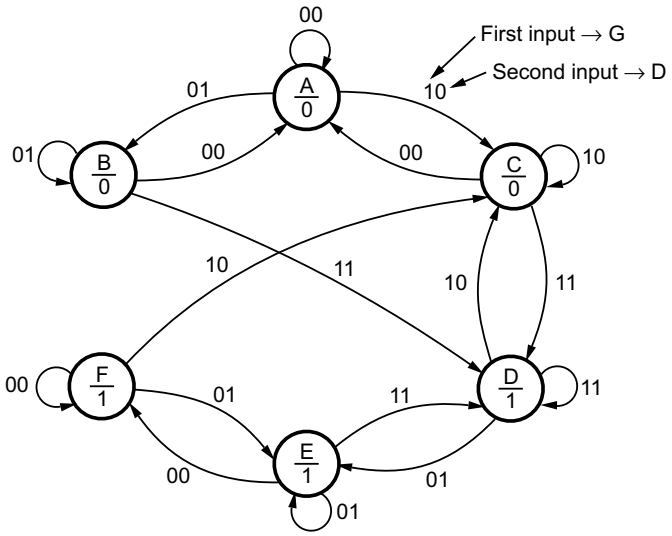


Fig. 9.6.45

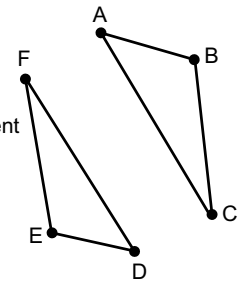
Step 2 : Reduction of primitive flow table

Present state	Next state for GD inputs				Output Q
	00	01	11	10	
A	(A)	B	**D**	C	0
B	A	(B)	D	–	0
C	A	**B**	D	(C)	0
D	**F**	E	(D)	C	1
E	F	(E)	D	**C**	1
F	(F)	E	**D**	C	1

Equivalent states (S_1)

Equivalent states (S_0)

(a)



(b) Merger graph

Fig. 9.6.46 Equivalent states

The merger graph shown in the Fig. 9.6.47 gives the two compatible pairs as a set of maximal compatibilities
(A, B, C) $\rightarrow S_0$ (D, E, F) $\rightarrow S_1$.

Present state	Next state for GD inputs				Output Q
	00	01	11	10	
S_0	(S_0)	(S_0)	S_1	(S_0)	0
S_1	(S_1)	(S_1)	(S_1)	S_0	1

Fig. 9.6.47 Reduced primitive flow table

Step 3 : State assignment

By making state assignment as $S_0 \rightarrow 0$ and $S_1 \rightarrow 1$, we can derive the transition table as shown in the Table 9.6.10.

Present state F	Next state for GD inputs				Output Q
	00	01	11	10	
0	0	0	1	0	0
1	1	1	1	0	1

Fig. 9.6.10

Step 4 : Realization of circuit using logic elements

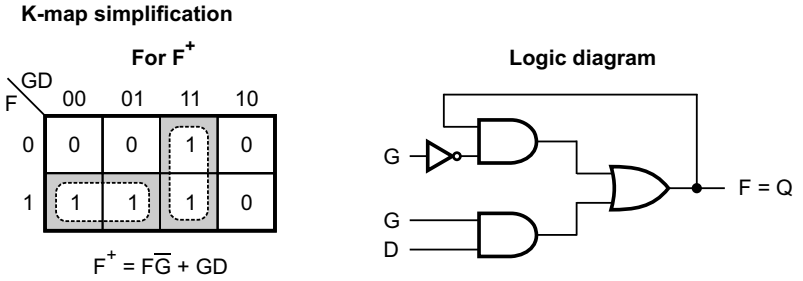


Fig. 9.6.48

Example 9.6.11 *Design a circuit with primary inputs A and B to give an output Z equal to 1 when A becomes 1 if B is already 1. Once Z = 1 it will remain so until A goes to 0. Draw waveform diagram, total state diagram, primitive flow table for designing this circuit.* **May-06**

Solution : The Fig. 9.6.49 shows the timing diagram for the given problem.

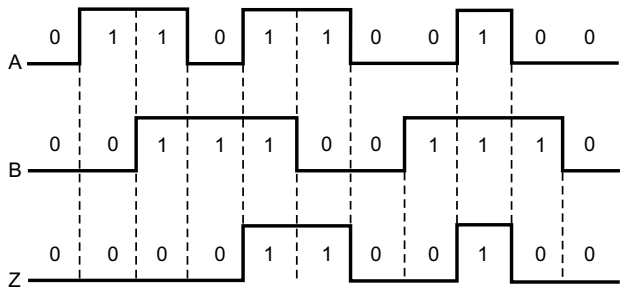


Fig. 9.6.49 Timing diagram

The state diagram for given problem statement can be given as shown in the Fig. 9.6.50.

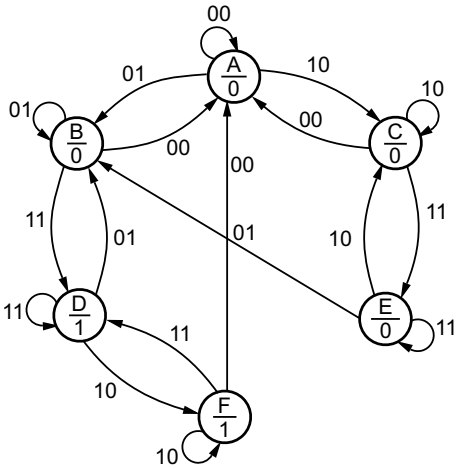


Fig. 9.6.50 Total state diagram

A primitive flow table is constructed from the state diagram as shown in the Fig. 9.6.51.

Present state	Next state for AB inputs				Output Z
	00	01	11	10	
A	(A)	B	–	C	0
B	A	(B)	D	–	0
C	A	–	E	(C)	0
D	–	B	(D)	F	1
E	–	B	(E)	C	0
F	A	–	D	(F)	1

Fig. 9.6.51

Review Questions

1. *Describe the steps involved in design of asynchronous sequential circuit in detail with an example.* **May-11, Marks 16**
2. *Define asynchronous sequential circuit, cycles, critical race, non-critical race.* **Dec.-08, Marks 8**

3. *When do you get the critical and non-critical races ? How will you obtain race free conditions ?* **Dec.-10, Marks 10**

4. *Race condition in asynchronous sequential circuits.* **Dec.-11, 12, Marks 6**

9.7 Data Synchronizers

Data synchronizer is a circuit that synchronizes a system where several modules are using their own clocks but no common system clock is available. It takes all asynchronous inputs and synchronize them; that is, make them consistent with the module clock.

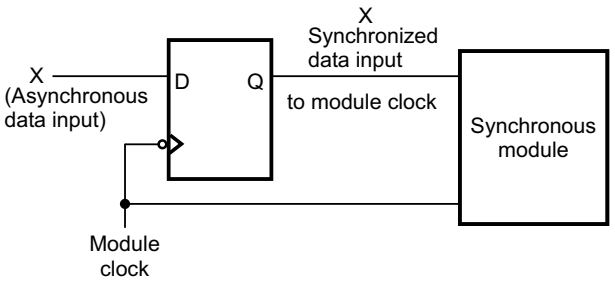


Fig. 9.7.1 Simple data synchronizer

The Fig. 9.7.1 shows the circuit for simple data synchronizer. It synchronizes single data input with the module clock. In this circuit, eventhough the asynchronous input B appears any time, it is applied to synchronous module in synchronization with module clock. The Fig. 9.7.2 shows the synchronizer timings.

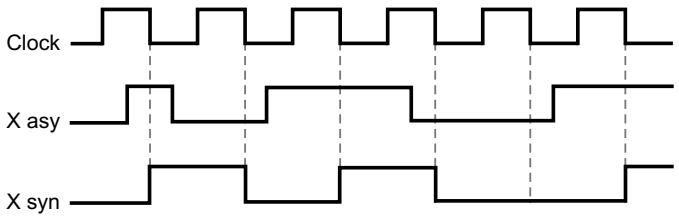


Fig. 9.7.2 Data synchronizer timings

Review Question

1. *Write a note on data synchronizer.*

9.8 Mixed Operating Mode Asynchronous Circuits

Dec.-06, May-07

We know that, critical **races** and hazards result from unequal propagation delay paths in the combinational logic that realizes the state variable excitation equations. A technique called **Mixed Operating Mode (MOM)** or **self-synchronization** is used to solve such timing problems in an asynchronous sequential circuits. In this approach each state variable is partitioned so that it has both synchronous and asynchronous inputs. Many flip-flops have both synchronous and asynchronous inputs on the same chip, as shown in the Fig. 9.8.1.

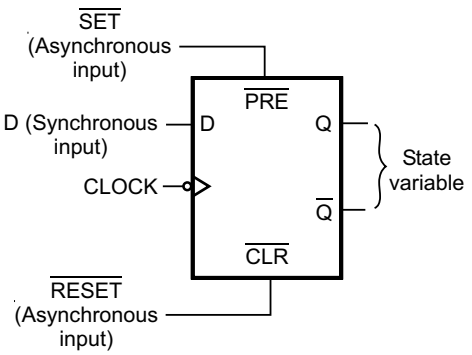


Fig. 9.8.1 Typical mixed mode flip-flop

By using both synchronous and asynchronous inputs of the same state variable, a problem due to essential hazards can be solved. If an essential hazard exists for a given state transition, then the synchronous inputs are used. If an essential hazard does not exist, then the asynchronous inputs are used. This approach is used by mixed operating mode. Here, extra logic is used to sense the existence of an essential hazard. If it exists then a clock pulse is generated and distributed to the specific state variable(s) that needs it. By providing such an **internal** clock pulse to only those variables where essential hazards may occur eliminates the critical races without slowing down all of the state transitions.

The Fig. 9.8.2 shows the mixed operating mode sequential circuit model. Here, synchronous and asynchronous inputs are separated and has separate excitation

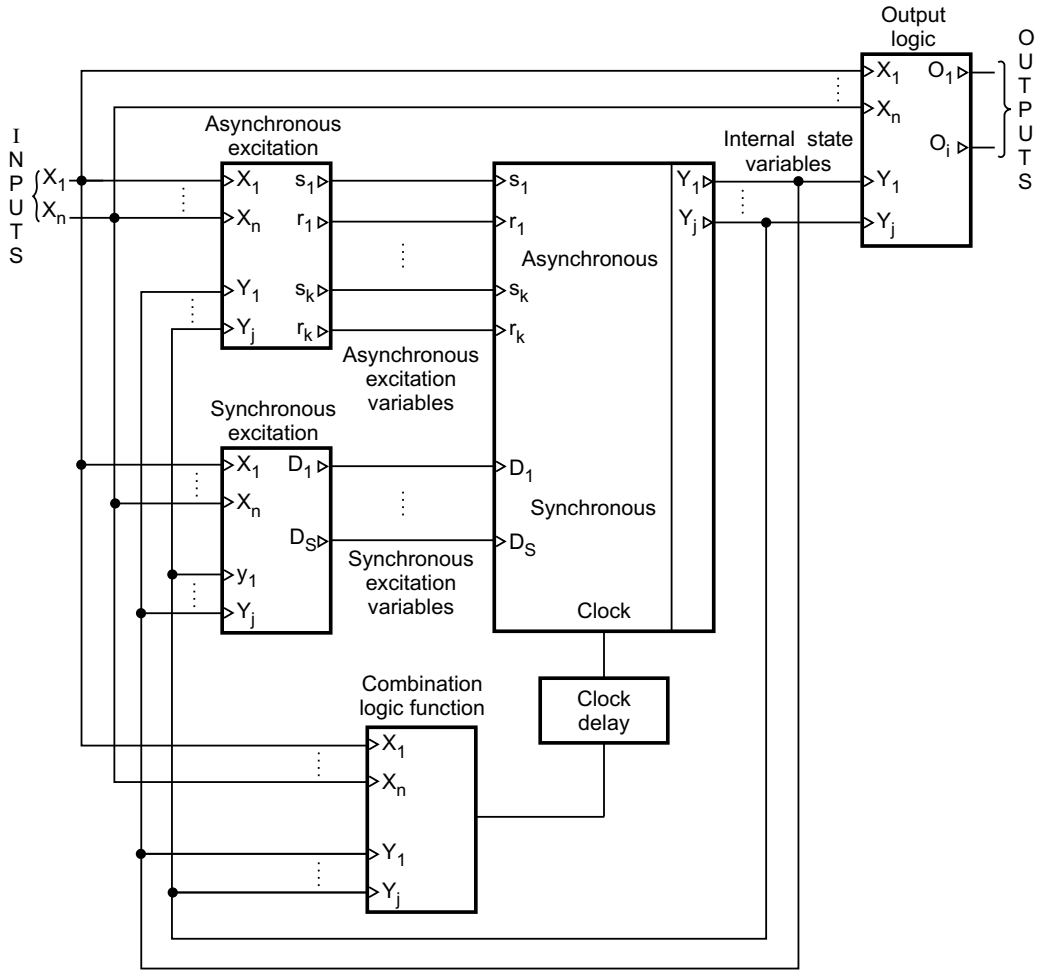


Fig. 9.8.2 Mixed operating mode sequential circuit model

functions. As shown in the Fig. 9.8.2, the asynchronous excitation inputs r_1 s_1, through r_k s_k, and the synchronous excitation inputs D_1 through D_s drive state variables Y_1 through Y_j. A separate combinational logic function is used to derive the clock pulses to the synchronous state variables from the system inputs X_1 through X_n and the state variables Y_1 through Y_j.

Review Questions

1. *Explain the mixed operating mode asynchronous circuits.*
2. *Draw and explain the mixed operating mode sequential circuit model.*
3. *Illustrate mixed operating mode sequential circuit model.* **Dec.-06, May-07, Marks 8**

9.9 Design of Hazard Free Switching Circuits

May-09, 10, 12, Dec.-08, 09, 11, 12

The unwanted switching transients (glitches) that may appear at the output of a circuit are called **Hazards**. The hazards cause the circuit to malfunction. The main cause of hazards is the different propagation delays at different paths. Hazards occur in the combinational circuits, where they may cause a temporary false output value. When such combinational circuits are used in the asynchronous sequential circuits, they may result in a transition to a wrong stable state.

There are two types of hazards : **Static hazards** and **dynamic hazards**. A static hazard exists if a signal is supposed to remain at particular logic value when an input variable changes its value, but instead the signal undergoes a momentary change in its required value. According to definition, the static hazards are further classified as **static-0** hazard and **static-1** hazard.

In a combinational circuit, if output goes momentarily 0 when it should remain a 1, the hazard is known as **static-1 hazard**. On the other hand, if output goes momentarily 1 when it should remain a 0, the hazard is known as **static-0 hazard**. Another type of hazard is **dynamic hazard** in which output changes three or more times when it should change from 1 to 0 or from 0 to 1. The Fig. 9.9.1 shows the three types of hazards.

(a) Static-1 hazard (b) Static-0 hazard (c) Dynamic hazard

Fig. 9.9.1 Types of hazards

The Fig. 9.9.2 shows circuit with hazards. Assume that, initially, inputs x_1 and x_3 = 0 and x_2 = 1. This causes the output of gate 1 to be 0, that of gate 2 to be 1, and the output of the circuit to be equal to 1. Now consider change in x_2 from

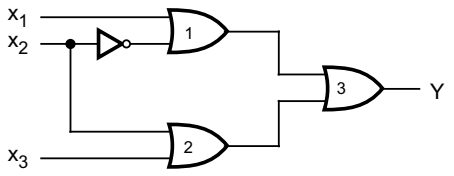


Fig. 9.9.2

1 to 0. The output of gate 1 changes to 1 and that of gate 2 changes to 0, leaving the output at 1. However, the output momentarily goes to 0 if the propagation delay through the inverter is taken into consideration. The delay in the inverter causes the output of gate 2 to change to 0 before the output of gate 1 changes to 1. In this situation, both inputs of gate 3 are momentarily equal to 0, causing the output to go to 0 for the short time equal to the propagation delay of the inverter. This is illustrated in the Fig. 9.9.3.

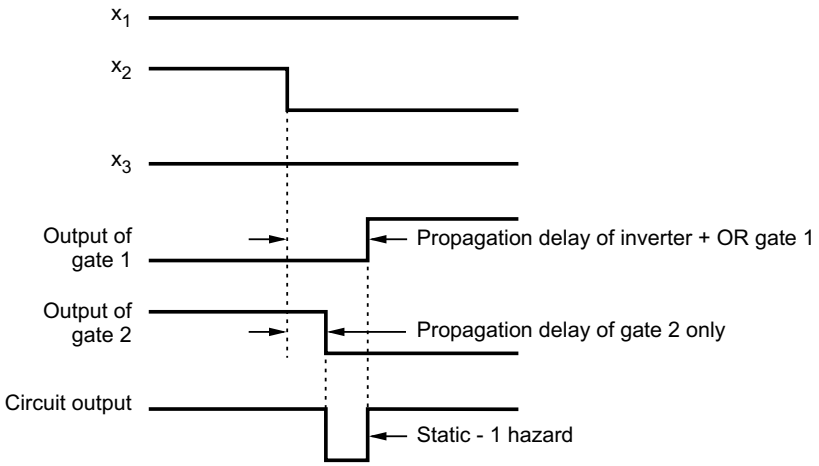


Fig. 9.9.3 Waveforms showing static-1 hazard

9.9.1 Eliminating a Hazard

The hazard exists because of the change of input results in a different product terms covering two minterms or different sum terms covering two maxterms. Whenever the circuit move from one product term to another or move one sum term to another, there is a possibility of a momentary interval when neither term is equal to 1, giving rise to an undesirable 0 output. Hazards can be eliminated by enclosing two minterms or maxterms in question. For example, if the circuit has minterms $x_1\ x_2 + \bar{x}_2 x_3$, then these two minterms must be enclosed by introducing another minterm x_1x_3. This is illustrated in Fig. 9.9.4.

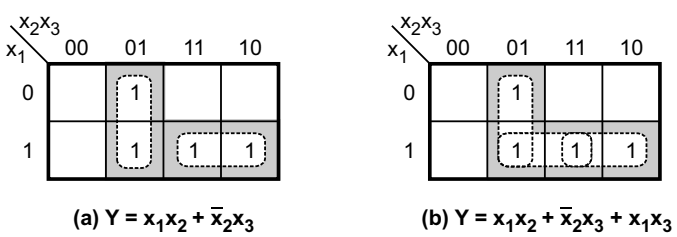


Fig. 9.9.4 Eliminating hazards

Example 9.9.1 *Give hazard-free realisation for the following Boolean function.*
$f(A, B, C, D) = \sum m(0, 2, 6, 7, 8, 10, 12)$

Solution : The given function can be implemented using K-map as shown in the Fig. 9.9.5 and Fig. 9.9.6 shows the additional product term, $\overline{A}C\overline{D}$ overlapping two groups (group 1 and group 2) for hazard free realization. Group 1 and group 3 are already overlapped hence they do not require additional minterm for grouping.

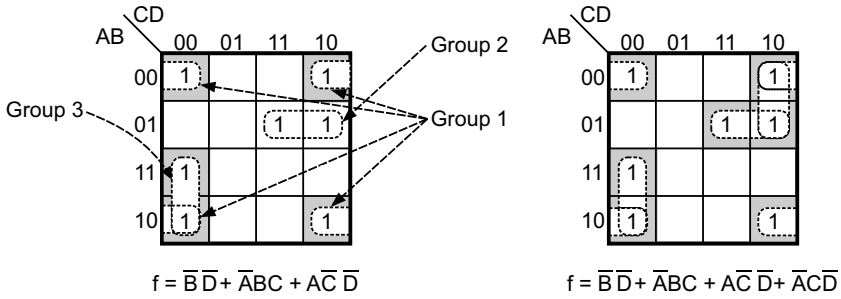


Fig. 9.9.5

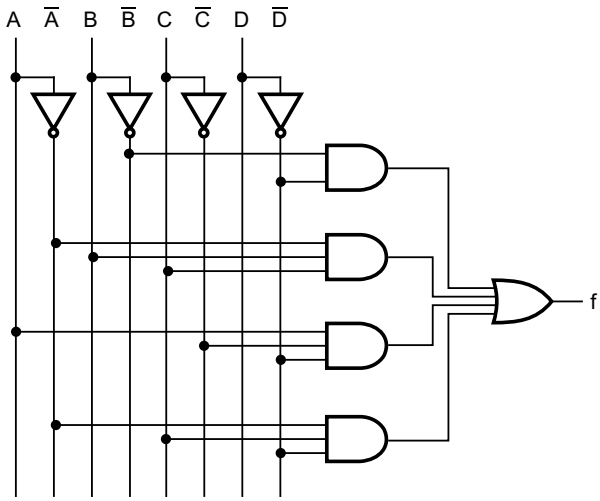


Fig. 9.9.6

9.9.2 Hazards in Sequential Circuits

We know that, in sequential circuits, the combinational circuits are associated with them to drive the flip-flop inputs. In synchronous sequential circuits, the hazards due to combinational circuits associated with them are not of concern. This is because momentary errorneous signals are not generally troublesome in synchronous circuits. However, if a momentary incorrect signal is fed back in an asynchronous sequential circuit, it may cause the circuit to go to the wrong stable state.

Let us consider the logic diagram and its transition table as shown in Fig. 9.9.7 and Fig. 9.9.8. For the circuit shown in Fig. 9.9.7, if the circuit is in total stable state YX_1X_2 = 111 and input X_2 changes from 1 to 0, the next total stable state should be YX_1X_2 = 110. However, because of hazard, the output Y may go to 0 momentarily. If this false signal feeds back into AND2 before the output of the inverter goes to 1, the output of AND2 will remain at 0 and the circuit will switch to the incorrect total stable state 010.

Following figure shows hazards in an asynchronous sequential circuit.

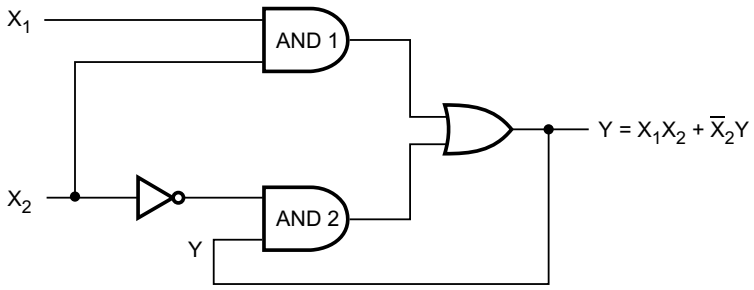


Fig. 9.9.7 Logic diagram

Y \ X_1X_2	00	01	11	10
0	(0)	(0)	1	(0)
1	(1)	0	(1)	(1)

Fig. 9.9.8 Transition table

Such a hazard can be eliminated by enclosing two minterms by another minterm as shown in the Fig. 9.9.9.

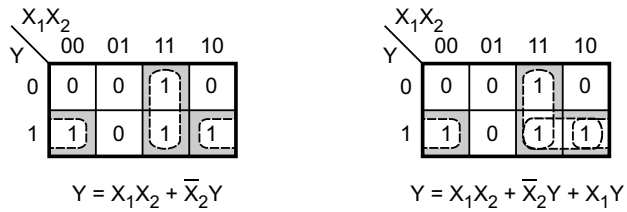


Fig. 9.9.9 The two minterms are enclosed by introducing another minterm X_1Y

Therefore, the hazard free asynchronous sequential circuit will be as shown in Fig. 9.9.10.

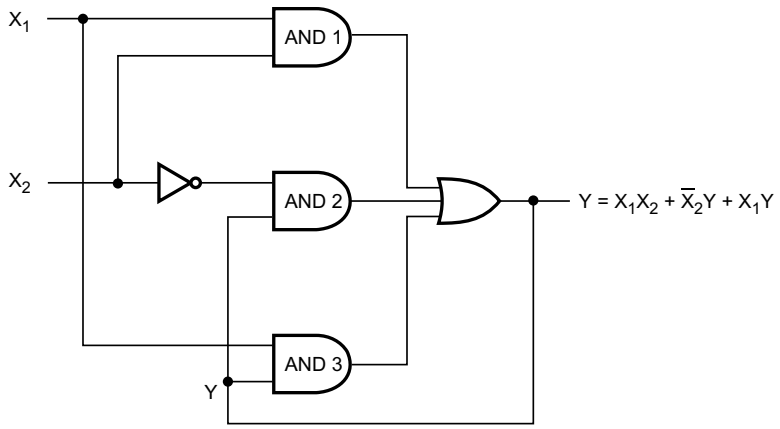


Fig. 9.9.10 Hazard free circuit

9.9.3 Essential Hazards

In the previous section we have seen static and dynamic hazards and remedies to remove it. There is another type of hazard that may occur in asynchronous sequential circuits, called **essential hazards**. An essential hazard is caused by unequal delays along two or more paths that originate from the same input. Such hazards can be eliminated by adjusting the amount of delays in the affected path.

9.9.4 Eliminating Essential Hazards

We can also avoid essential hazards in asynchronous sequential circuits by implementing them using SR latches. A momentary 0 signal applied to the S or R inputs of a NOR latch will have no effect on the state of the circuit. Similarly, a momentary 1 signal applied to the S and R inputs of a NAND latch will have no effect on the state of the latch.

Let us consider a NAND SR latch with the following Boolean functions for S and R.

$$S = AB + CD$$

$$R = \overline{A}C$$

Such circuit can be implemented using two-level circuit of NAND gates as shown in the Fig. 9.9.11. The first level consists of NAND gates that implement each product term in the original Boolean expression of S and R. The second level forms the cross-coupled connection of the SR latch with inputs that come from the outputs of each NAND gate in the first level.

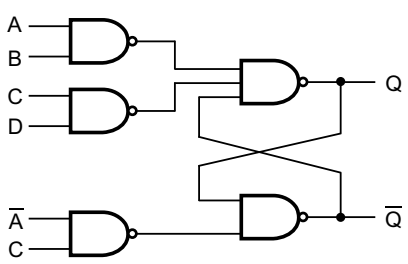
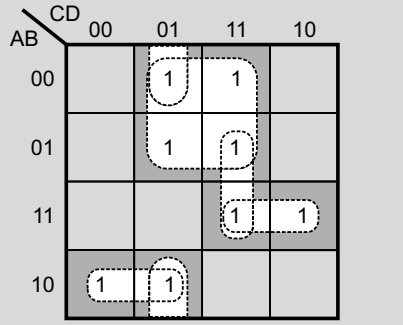


Fig. 9.9.11 Implementation of latch

Illustrative Examples

Example 9.9.2 *Implement the switching function* $F = \sum (1, 3, 5, 7, 8, 9, 14, 15)$ *by a static hazard free two level AND-OR gate network.* **Dec.-02, CSE, Marks 16**



$F = A\overline{B}\,\overline{C} + \overline{B}\,\overline{C}D + \overline{A}D + BCD + ABC$

Fig. 9.9.12

Solution : AND-OR Network

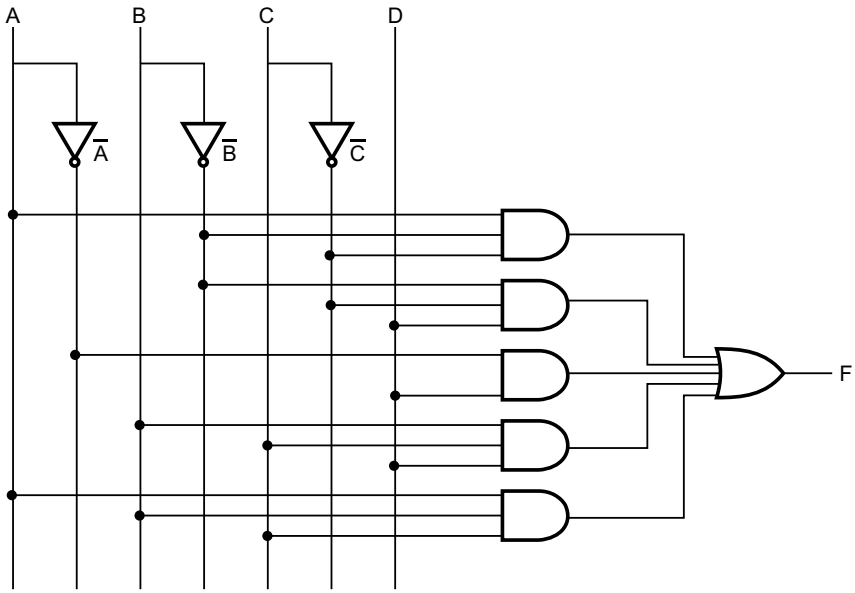


Fig. 9.9.13

Example 9.9.3 *Using Unger's theorem, show that the following machine has essential hazard.*

Dec.-02, CSE, Marks 16

Solution :

PS	NS	
	x = 0	x = 1
00	00	01
01	11	01
10	00	10
11	11	10

We have seen that, an essential hazard exists in an asynchronous circuit when a present stable total state, S has a different next total states after one transition of an input variable X_i and after three transitions of an input variable X_i.

1. $00(0) \rightarrow \mathbf{01(1)} \rightarrow 11(0) \rightarrow \mathbf{10(1)}$ essential hazard
2. $01(1) \rightarrow \mathbf{11(0)} \rightarrow 10(1) \rightarrow \mathbf{00(0)}$ essential hazard
3. $10(1) \rightarrow \mathbf{00(0)} \rightarrow 01(1) \rightarrow \mathbf{11(0)}$ essential hazard
4. $11(0) \rightarrow \mathbf{10(1)} \rightarrow 00(0) \rightarrow \mathbf{01(1)}$ essential hazard

The essential hazard are

1. $00(0) \rightarrow 01(1)$
2. $01(1) \rightarrow 11(0)$
3. $10(1) \rightarrow 00(0)$
4. $11(0) \rightarrow 10(1)$

Example 9.9.4 *Implement the switching function* $F = \sum (0, 1, 3, 4, 8, -12)$ *by a static hazard free two level OR-AND gate network.*

Solution :

$$\overline{F} = \overline{C}\,\overline{D} + A\overline{B} + \overline{B}D$$

$$= \overline{\overline{F}} = \overline{\overline{C}\,\overline{D} + A\,\overline{B} + \overline{B}D}$$

$$= \overline{(C+D)\left(\overline{A}+B\right)\left(B+\overline{D}\right)}$$

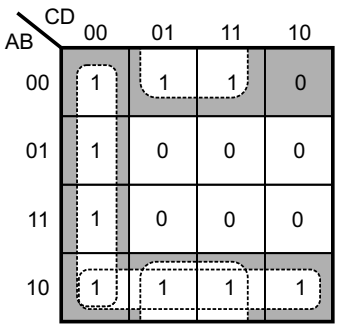


Fig. 9.9.14

Logic diagram

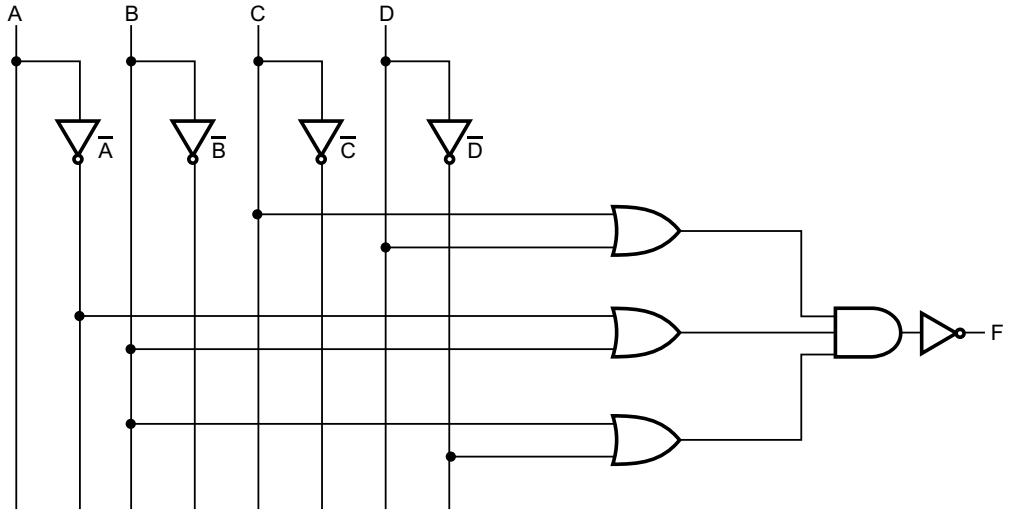


Fig. 9.9.15

Example 9.9.5 *Show that no static 0 (static 1) hazard can happen in a two level AND-OR (OR-AND) realisation of a switching function F.* **May-03, CSE, Marks 16**

Solution : Dynamic hazards due to a change in an input variable x_i can only occur if there are three or more paths between the x_i (and/or x_i') input and the network output. Such condition does not exists in two level AND-OR gate networks hence dynamic hazards do not occur in two level AND-OR gate networks.

Example 9.9.6 *Find a static and dynamic hazard free realization for the following function using i) NAND gates ii) NOR gates F (a, b, c, d) = $\sum m$ (1, 5, 7, 14, 15)*

May-04, CSE, Marks 16

Solution : i) Circuit realization using NAND gate

$$F = \overline{a}\,\overline{c}\,d + \overline{a}\,b\,d + b\,c\,d + a\,b\,c$$

ii) Circuit realization using NOR gate

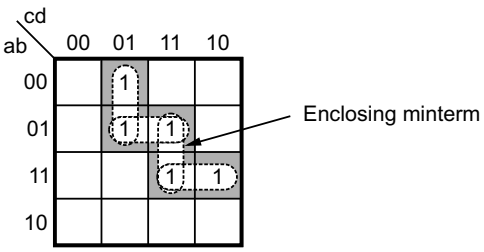


Fig. 9.9.16

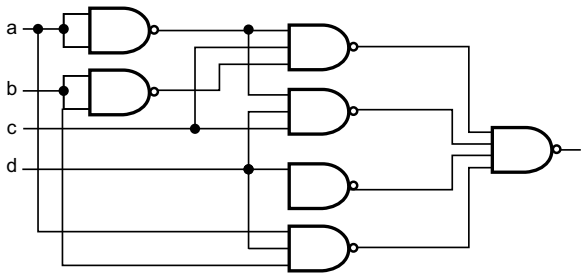


Fig. 9.9.17

Steps :

- Complement inputs.
- Complement output and
- Replace NAND gates by NOR gates.

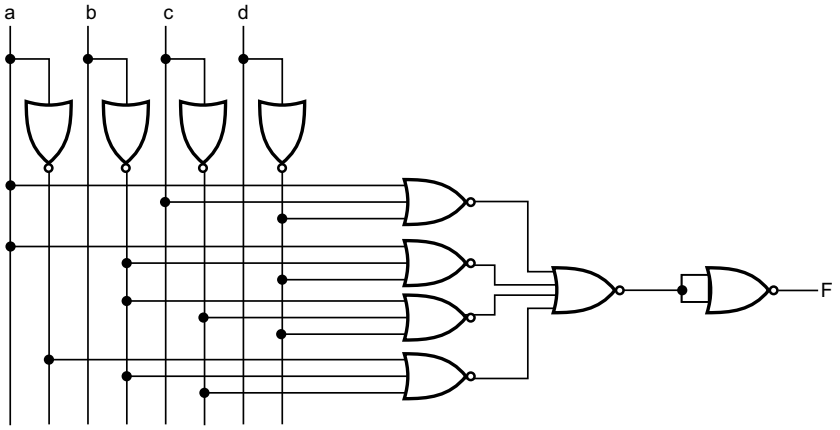


Fig. 9.9.18

Example 9.9.7 *Give hazard-free realization for the following Boolean functions.*

$f(A, B, C, D) = \sum m(0, 1, 5, 6, 7, 9, 11)$ **Dec.-07, CSE/IT, Marks 8**

Solution :

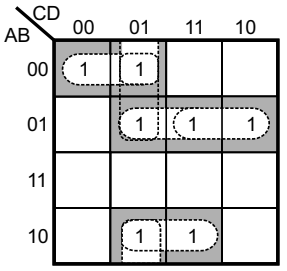


Fig. 9.9.19

$$F = \overline{A}\,\overline{B}\,\overline{C} + A\,\overline{B}\,D + \overline{B}\,\overline{C}\,D + \overline{A}\,\overline{C}\,D + \overline{A}\,B\,D + \overline{A}\,B\,C$$

Example 9.9.8 *Give hazard-free realization for the following Boolean function.*

$F(I, J, K, L) = \sum m(1, 3, 4, 5, 6, 7, 9, 11, 15)$ **May-08, Marks 8**

Solution :

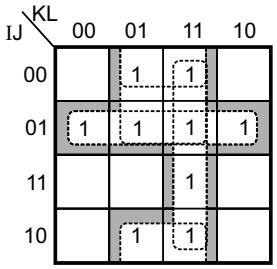


Fig. 9.9.20

$$F = \bar{J}L + \bar{I}J + KL + \bar{I}L$$

Review Questions

1. *What are hazards ?* **Dec.-09, Marks 2**
2. *Define static hazard. How it can be avoided ?* **May-09, Marks 2**
3. *Explain dynamic hazard.* **Dec.-08, Marks 2**
4. *What is a hazard ? Explain the different types of hazards. What is an essential hazard ? Discuss in detail how hazards can be eliminated.* **Dec.-08, 12, Marks 16**
5. *Briefly explain the dynamic and essential hazards.* **May-09, Marks 8**
6. *What are hazard free digital circuits?* **May-10, Marks 2**
7. *What are called as essential hazards? How does the hazard occur in sequential circuits? How can the same be eliminated using SR latches? Give an example.* **May-10, Marks 16**
8. *Guidelines for hazard free circuit design.* **Dec-11, Marks 4**
9. *Write a note on hazards in asynchronous sequential circuits.* **Dec.-11,12, Marks 6**
10. *Write a short note on hazard free switching circuits.* **May-12, Marks 8**

9.10 Two Marks Questions with Answers

Q.1 ***What is an asynchronous sequential circuit ?*** **Dec.-04**

Ans. : The sequential circuits in which the change in input signals can affect memory element at any instant of time are called asynchronous sequential circuits.

Q.2 ***How does the operation of an asynchronous input differ from that of a synchronous input ?*** **Dec.-05**

Ans. : In synchronous sequential circuits, memory elements are clocked flip-flops. Hence input signals can affect the memory elements only at discrete instants of time.

In asynchronous sequential circuits, memory elements are either unclocked flip-flops or time delay elements. Therefore in asynchronous sequential circuits change in input signals can affect memory element at any instant of time.

Q.3 *What are the types of asynchronous circuits ?*

Ans. : 1. Fundamental mode circuits

2. Pulse mode circuits.

Q.4 *What is a fundamental mode asynchronous sequential circuit ?*

May-03, Dec.-03

Ans. : According to how input variables are to be considered, fundamental mode circuit assumes that :

- The input variables change only when the circuit is stable.
- Only one input variable can change at a given time and
- Inputs are levels and not pulses.

Q.5 *What is pulse mode circuit ?*

Ans. : Refer section 9.2.

Q.6 *Define secondary variables and excitation variables.*

Ans. : The present state and next state variables in asynchronous sequential circuits are called **secondary variables** and **excitation variables**, respectively.

Q.7 *Define flow table in asynchronous sequential circuit.* **May-07, Dec.-06, 07**

Ans. : In asynchronous sequential circuit state table is known as flow table because of the behaviour of the asynchronous sequential circuit. The stage changes occur in independent of a clock, based on the logic propagation delay, and cause the states to flow from one to another.

Q.8 *Define primitive flow table.* **Dec.-08**

Ans. : It is defined as a flow table which has exactly one stable state for each row in the table. The design process begins with the construction of primitive flow table.

Q.9 *What are the steps for the design of asynchronous sequential circuit?*

Ans. : Refer sections 9.4 and 9.6.

Q.10 *Define merger graph.*

Ans. : The merger graph is defined as follows. It contains the same number of vertices as the state table contains states. A line drawn between the two state vertices indicates each compatible state pair. It two states are incompatible no connecting line is drawn. It is used as a tool in state reduction process.

Q.11 *What is a cycle ? or When does a cycle occur ?* **May-03, 05**

Ans. : A cycle occurs when an asynchronous circuit makes a transition through a series of unstable states. The cycle does not contain a stable state, the circuit will go from one unstable to stable to another, until the inputs are changed.

Q.12 *What are races ?* **May-07, Dec.-06, 07, 08**

Ans. : When two or more binary state variables change their value in response to a change in an input variable, race condition occurs in an asynchronous sequential circuit. In case of unequal delays, a race condition may cause the state variables to change in an unpredictable manner.

Q.13 *Define noncritical race.*

Ans. : If the final stable state that the circuit reaches does not depend on the order in which the state variable changes, the race condition is not harmful and it is called a non critical race.

Q.14 *Define critical race ?* **May-06, Dec.-07, 10**

Ans. : If the final stable state depends on the order in which the state variable changes, the race condition is harmful and it is called a critical race.

Q.15 *What are the significance of state assignment ?*

Ans. : In synchronous circuits-state assignments are made with the objective of circuit reduction. In asynchronous circuits its objective is to avoid critical races.

Q.16 *What are the different techniques used in state assignment ?*

Ans. : 1. Shared row state assignment

2. One hot state assignment.

Q.17 *What are hazards ?* **Dec.-09**

Ans. : The unwanted switching transients (glitches) that may appear at the output of a circuit are called **Hazards**.

Q.18 *What are the two types of hazards ?*

Ans. : The two types of hazards are :

- Static hazard
- Dynamic hazard.

Q.19 *What is static hazard ?*

Ans. : A static hazard exists if a signal is supposed to remain at particular logic value when an input variable changes its value, but instead the signal undergoes a momentary change in its required value.

Q.20 *What are static-0 and static-1 hazards ?*

Ans. : In a combinational circuit, if output goes momentarily 0 when it should remain a 1, the hazard is known as **static-1 hazard**. On the other hand, if output goes momentarily 1 when it should remain a 0, the hazard is known as **static-0 hazard**.

Q.21 *Explain dynamic hazard.* **Dec.-08**

Ans. : The hazard in which output changes three or more times when it should change from 1 to 0 or from 0 to 1 is called dynamic hazard.

Q.22 *What is the cause of essential hazard ?* **May-03, Dec.-04**

Ans. : An essential hazard is caused by unequal delays along two or more paths that originate from the same input. Such hazards can be eliminated by adjusting the amount of delays in the affected path.

❑❑❑

10 Memory Devices and PLDs

Syllabus

Classification of memories - ROM - ROM organization - PROM - EPROM - EEPROM - EAPROM, RAM - RAM organization - Write operation - Read operation-Memory cycle - Timing wave-forms - Memory decoding - Memory expansion - Static RAM cell - Bipolar RAM cell - MOSFET RAM cell - Dynamic RAM cell - Programmable logic devices - Programmable Logic Array (PLA) - Programmable Array Logic (PLA) - Field Programmable Gate Arrays (FPGA) - Implementation of combinational logic circuits using ROM, PLA, PAL.

Contents

10.1 Introduction . . . . . **Dec.-04, 05, 08, May-06,** · · Marks 16

10.2 Classification of Memory **May-06,07,09, 12, Dec.-05, 06, 12**

10.3 ROM (Read Only Memory) **May-03, 05, 06, 07, 09, 11,**
. . . . . **Dec.-03, 06, 07, 08** · · · · · · · Marks 10

10.4 ROM Organization . . . . . **Dec.-06, May-08** · · · · · · · · · Marks 8

10.5 RAM (Random Access Memories) **Dec.-03, 05, 06, 07, 08, 09, 12,**
. . . . . **May-06, 09, 10, 12,** · · · · · · · Marks 16

10.6 RAM Organization . . . . . **Dec.-05,** · · · · · · · · · · · · · · · Marks 12

10.7 DRAM Organization . . . . . **Dec.-11,** · · · · · · · · · · · · · · · · Marks 8

10.8 Memory Cycles and Timing Waveforms . . . . . **May-08, 09, Dec.-08, 10,** · · · Marks 8

10.9 Memory Decoding **Dec.-03, 08, 12,**
. . . . . **May-07, 12,** · · · · · · · · · · · · · Marks 10

10.10 Memory Expansion . . . . . **May-10, 12, Dec.-12,** · · · · · · Marks 8

10.11 Programmable Logic Devices (PLDs) **May-03, 05, 07, 09, 11,**
. . . . . **Dec.-03, 06, 09, 12,** · · · · · · Marks 16

10.12 Field Programmable Gate Arrays (FPGA). **May-06, 07, 08, 09, 10, 12,**
. . . . . **Dec.-04, 08, 10,** · · · · · · · · · · Marks 8

10.13 Two Marks Questions and Answers

10.1 Introduction

Dec.-04, 05, 08, May-06

Memories are made up of registers. Each register in the memory is one storage location also called **memory location**. Each memory location is identified by an **address**. The number of storage locations can vary from a few in some memories to hundreds of thousand in others. Each location can accommodate one or more bits. Generally, the total number of bits that a memory can store is its **capacity**. Most of the types the capacity is specified in terms of bytes (group of eight bits).

Each register consists of storage elements (flip-flops or capacitors in semiconductor memories and magnetic domain in magnetic storage), each of which stores one bit of data. A storage element is called a **cell**.

The data stored in a memory by a process called **writing** and are retrieved from the memory by a process called **reading**. Fig. 10.1.1 illustrates in a very simplified way the concept of write, read, address and storage capacity for a generalized memory.

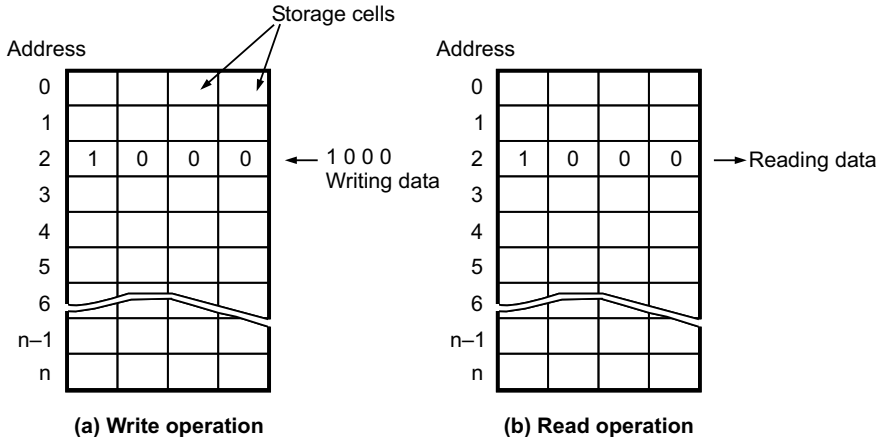


Fig. 10.1.1

As shown in the Fig. 10.1.1 a memory unit stores binary information in groups of bits called **words**. A word in memory is an entity of bits that moves in and out of storage as a unit. A word having group of 8-bits is called a **byte**. Most computer memories use words that are multiples of eight bits in length. Thus, a 16-bit word contains two bytes, and a 32-bit word is made of 4-bytes.

The communication between a memory and its environment is achieved through data lines, address selection lines, and control lines that specify the direction of transfer. The Fig. 10.1.2 shows the block diagram of memory unit. Then data lines provide the information to be stored in memory and the k address lines

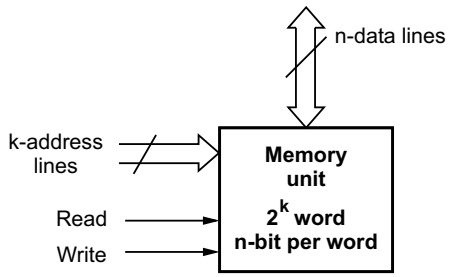


Fig. 10.1.2 Block diagram of memory unit

specify the particular word chosen among the many available. The two control inputs specify the direction transfer.

When there are k address lines we can access 2^k memory words. For example, if k = 10 we can access 2^{10} = 1024 memory words.

Illustrative Examples

Example 10.1.1 *A bipolar RAM chip is arranged as 16 words. How many bits are stored in the chip ?* **Dec.-05, Marks 4**

Solution : $16 \times 8 = 128$ bits $\because$ one word = 8 bits.

Example 10.1.2 *How many address bits are needed to operate a 2K × 8 ROM?*

Solution : 2K memory locations = 2048 locations

Since $2^{11} = 2048$, we need 11 address lines.

Example 10.1.3 *How many locations are addressed using 18 address bits?* **Dec.-08, Marks 2**

Solution : The number of locations addressed = $2^{18} = 262144$.

Review Questions

1. *Define a memory cell. Give an example.* **Dec.-04, Marks 2**
2. *Define a 'memory location' and a 'cell'.* **May-06, Marks 2**
3. *Write a descriptive note on memories.* **May-06, Marks 16**

10.2 Classification of Memory

May-06, 07, 09, 12, Dec.-05, 06, 12

The Table 10.2.1 shows the classification of semiconductor memory devices. The semiconductor memory devices can be categorized in several ways according to their functional and architectural characteristics.

Classification of semiconductor memories			
Non-volatile memory		**Volatile memory**	
Read Only Memory (ROM)	**Read/Write Memory (NVRAM)**	**Read/Write Memory (RWM)**	
■ Mask-Programmable ROM	■ EPROM	**Random Access**	**Non-Random Access**
■ Programmable ROM	■ EEROM	■ SRAM	■ FIFO
	■ FLASH	■ DRAM	■ LIFO
			■ Shift Register

Note :

- ROM : Read Only Memory
- PROM : Programmable ROM
- EPROM : Erasable PROM
- EEPROM : Electrically Erasable PROM
- SRAM : Static Random Access Memory
- DRAM : Dynamic Random Access Memory
- FIFO : First-in First-out
- LIFO : Last-in First-out

Table 10.2.1 Classification of semiconductor memories

Boardly semiconductor memories are classified as volatile memories and non-volatile memories. Volatile memories can retain their state as long as power is applied. On the other hand, non-volatile memories can hold data even if power is turned off. Read/Write Memories (RWMs) are those memories, which allows both read and write operations. They are used in applications where data has to change continuously. They are also used for temporary storage of data. ROM memories allow only read operation. They are used to store monitor programs and constants used in the program.

The volatile memories which can hold data as long as power is ON are called **static RAMs (SRAMs).** Dynamic RAMs (DRAMs) stores the data as a charge on the capacitor and they need refreshing of charge on the capacitor after every few milliseconds to hold the data even if power is ON.

EPROM and EEPROM are erasable memories in which the stored data can be erased and new data can be stored.

The semiconductor memories are also classified as Bipolar and MOS memories depending upon the type of transistors used to construct the individual cell.

Review Questions

1. *Given the classification of semiconductor memories.* **May-09, Dec.-12, Marks 8**
2. *Which memory is called volatile ? Why ?* **Dec.-05, Marks 2**
3. *What is meant by 'static' and 'dynamic' memories ?* **May-06, Marks 2**
4. *Explain static memory.* **Dec.-06, May-07, Marks 2**
5. *Discuss the classification of ROM and ROM memories.* **May-12, Marks 8**

10.3 ROM (Read Only Memory) May-03, 05, 06, 07, 09, 11, Dec.-03, 06, 07, 08

We can't write data in read only memories. It is non-volatile memory i.e. it can hold data even if power is turned off. Generally, ROM is used to store the binary codes for the sequence of instructions you want the computer to carry out and data such as look up tables. This is because this type of information does not change.

It is important to note that although we give the name RAM to static and dynamic read/write memory devices, that does not mean that the ROMs that we are using are also not random access devices. In fact, most ROMs are accessed randomly with unique addresses.

The Fig. 10.3.1 shows the typical configuration of a ROM cell. It consists of a transistor T and switch P. The transistor T is driven by the word line. The contents of cell can be read from the cell when word line is logic 1. A logic value 0 is read if the transistor is connected to ground through switch P. If switch P is open, a logic value 1 is read. The bit line is connected through a resistor to the power supply. A sense circuit at the end of the bit line generates the proper output value. Data is stored into a ROM when it is manufactured.

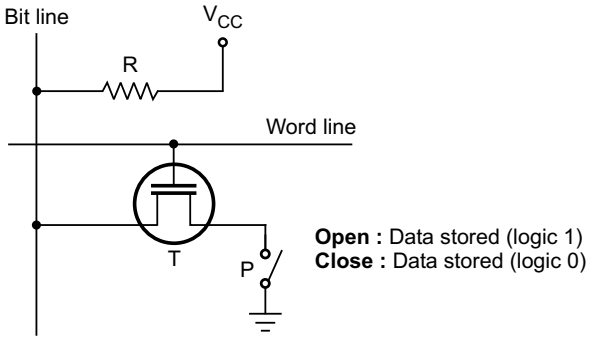


Fig. 10.3.1 ROM cell

There are four types of ROM : Masked ROM, PROM, EPROM and EEPROM or E^2PROM.

10.3.1 PROM (Programmable Read Only Memory)

PROMs are programmed by user. To provide the programming facility, each address select and data line intersection has its own fused MOSFET or transistor. When the fuse is intact, the memory cell is configured as a logic 1 and when fuse is blown (open circuit), the memory cell is logical 0. Logical 0s are programmed by selecting the appropriate select line and then driving the vertical data line with a pulse of high current. The Fig. 10.3.2 shows a PROM fused MOSFET memory cell.

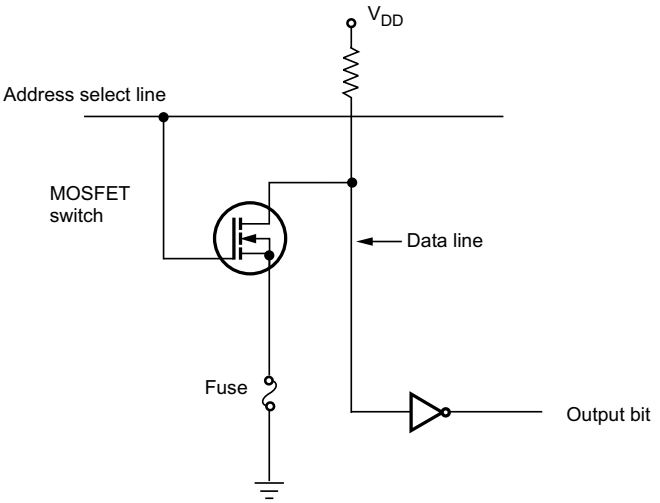


Fig. 10.3.2 Single fused PROM cell

Fig. 10.3.3 shows four byte PROM. It has diodes in every bit position; therefore, the output is initially all 0s. Each diode, however has a fusible link in series with it. By addressing bit and applying proper current pulse at the corresponding output, we can blow out the fuse, storing logic 1 at that bit position. The fuse uses material like nichrome and polycrystalline. For blowing the fuse it is necessary to pass around 20 to 50 mA of current for period 5 to 20 µs. The blowing of fuses according to the truth table is called programming of ROM. The user can program PROMs with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

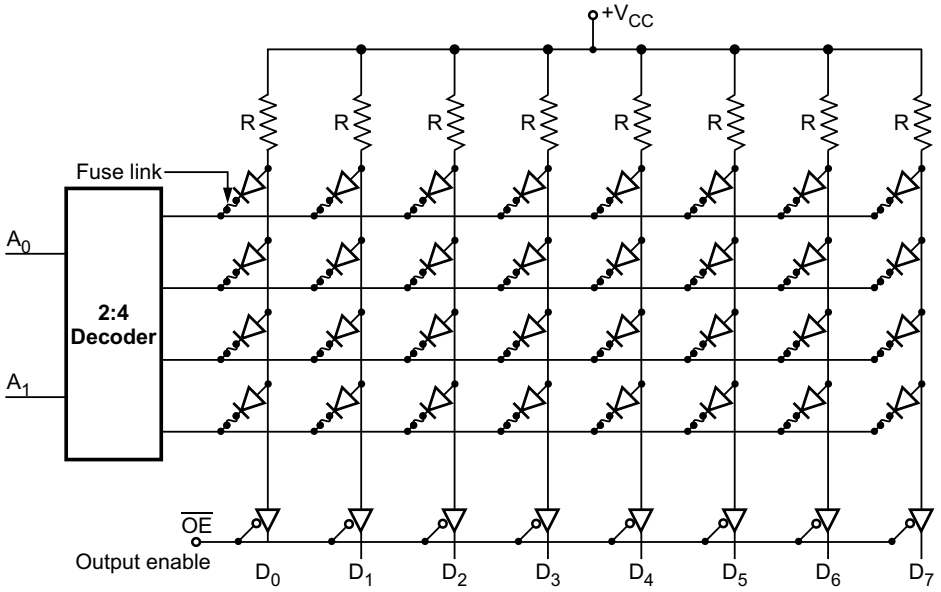


Fig. 10.3.3 Four byte PROM

10.3.2 EPROM (Erasable Programmable Read Only Memory)

Erasable programmable ROMs use MOS circuitry. They store 1's and 0's as a packet of charge in a buried layer of the IC chip. EPROMs can be programmed by the user with a special EPROM programmer. The important point is that we can erase the stored data in the EPROMs by exposing the chip to ultraviolet light through its quartz window for 15 to 20 minutes, as shown in the Fig. 10.3.4.

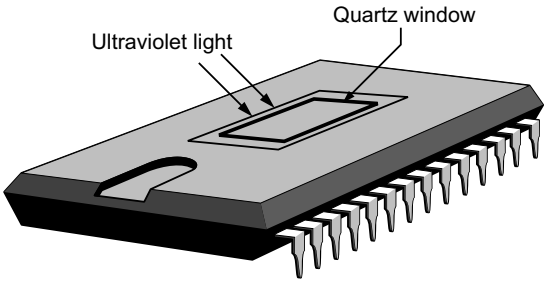


Fig. 10.3.4 EPROM

It is not possible to erase selective information, when erased the entire information is lost. The chip can be reprogrammed. This memory is ideally suitable for product development, experimental projects and college laboratories, since this chip can be reused many times, over.

EPROM Programming :

When erased each cell in the EPROM contains 1. Data is introduced by selectively programming 0's into the desired bit locations. Although only 0's will be programmed, both 1's and 0's can be presented in the data.

During programming address and data are applied to address and data pins of the EPROM. When the address and data are stable, program pulse is applied to the program input of the EPROM. The program pulse duration is around 50 ms and its amplitude depends on EPROM IC. It is typically 5.5 V to 25 V. In EPROM, it is possible to program any location at any time - either individually, sequentially, or at random.

10.3.3 EEPROM (Electrically Erasable Programmable Read Only Memory)

Electrically erasable programmable ROMs also use MOS circuitry very similar to that of EPROM. Data is stored as charge or no charge on an insulated layer or an insulated floating gate in the device. The insulating layer is made very thin ($< 200\,\mathring{A}$). Therefore, a voltage as low as 20 to 25 V can be used to move charges across the thin barrier in either direction for programming or erasing. EEPROM allows selective erasing at the register level rather than erasing all the information since the information can be changed by using electrical signals. The EEPROM memory also has a special chip erase mode by which entire chip can be erased in 10 ms. This time is quite small as compared to time required to erase EPROM and it can be erased and reprogrammed with device right in the circuit. However, EEPROMs are most expensive and the least dense ROMs.

Review Questions

1. *Write note on ROM technologies.*
2. *Give the features of a ROM cell.*
3. *Briefly explain any four nonvolatile memory in detail.*
4. *Write a note on ROM.* **Dec.-03, Marks 4**
5. *Explain EPROM.*
6. *Write a note on EPROM.* **May-03, 05, 09, Marks 4**
7. *Mention the two types of erasable PROM.* **Dec.-06, Marks 2**
8. *How is individual location in a EEPROM programmed or erased ?* **May-06, Marks 2**
9. *Elaborate the single fused PROM cell with clear sketch.* **Dec.-06, Marks 6**

10. Write the advantages of EPROM over PROM.	**May-07, Marks 2**
11. Whether ROM is classified as nonvolatile storage device? Why?	**May-07, Marks 2**
12. Write a note on types of ROMs.	**Dec.-07, Marks 4**
13. Explain the various ROM organizations and give the uses for each type.	**Dec.-08, Marks 10**
14. Write a note on EEPROM.	**May-09, Marks 4**
15. Briefly explain the EPROM and EEPROM technology.	**May-11, Marks 6**

10.4 ROM Organization

Dec.-06, May-08

First, we will see the simple ROM. Fig. 10.4.1 shows a very simple four byte diode ROM. Diode ROM consists of only diodes and a decoder. As shown in the Fig. 10.4.1 address lines A_0 and A_1 are decoded by 2 : 4 decoder and used to select one of the four rows. As decoder output is active low, it places a logic 0 on the selected row. Each output data line goes to logic 0 if a diode connects the output data column to the selected row. Data is available on the output data lines only when output enable ($\overline{OE}$) signal is low. Table 10.4.1 shows the contents of ROM at four locations.

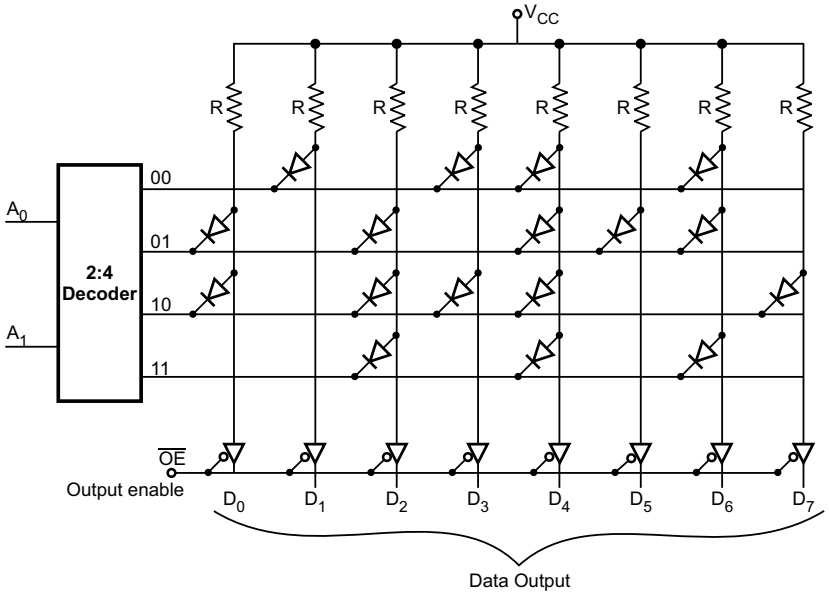


Fig. 10.4.1 Simple four byte diode ROM

Now a days ROMs use MOS technology instead of diode. Fig. 10.4.2 shows four nibble (half-byte) ROM using MOS transistors. Here, diodes and pull up resistors are replaced by MOS transistors. The address on the address lines (A_0 and A_1) is decoded by 2 : 4 decoder. Decoder selects one of the four rows making it logic 0. The inverter connected at the output of decoder inverts the state of selected row (i.e. logic 1). Therefore, each output data line goes to logic 0 if a gate of MOS transistor is connected

to row select lines. When gate of the MOS transistor is connected to the selected row, MOS transistor is turned on. This pulls the corresponding column data line to logic 0.

Address in binary	Binary Data								Data in Hex
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
00	1	0	1	0	0	1	0	1	A5
01	0	1	0	1	0	0	0	1	51
10	0	1	0	0	0	1	1	0	46

Table 10.4.1 Contents of ROM

In integrated circuits, a thin metallized layer connects the gates of some transistors to the row select lines. The gate connections of MOS transistors depend on the data to be stored in the ROM. Therefore, according to the user truth table, manufacturer can deposit thin layer of metal to connect gates of the transistors. Once the pattern/mask is decided, it is possible to make thousands of such ROMs. Such ROMs are called Mask-programmed ROMs. Masked ROMs are used in microprocessor based toys, TV games, home computers and other such high volume consumer products.

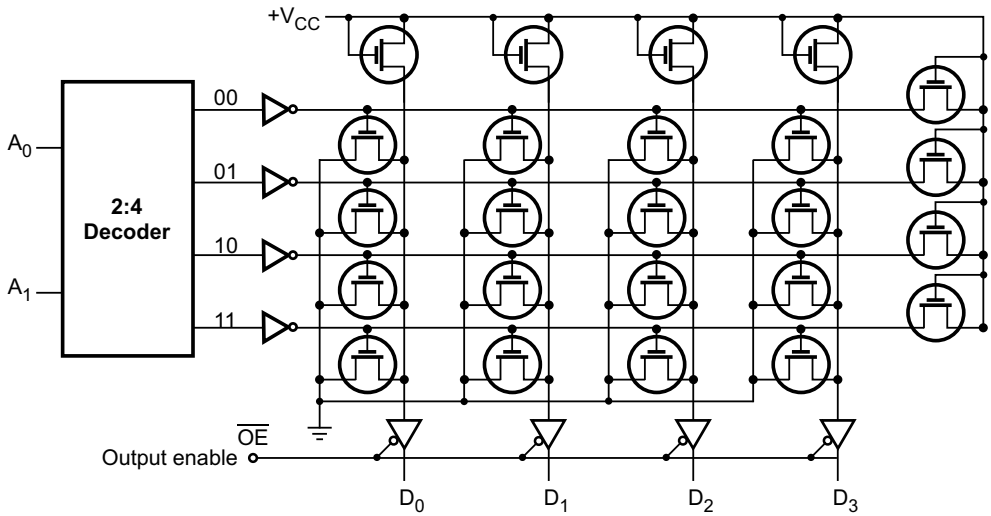


Fig. 10.4.2 Simple four half-byte ROM

Review Questions

1. *Describe the typical ROM internal organization with necessary diagram.* **Dec.-06, Marks 8**
2. *Illustrate the concept of 16 × 8-bit ROM arrangement with diagram.* **Dec.-06, Marks 8**
3. *Design a 16-bit ROM array and explain the operation.* **May-08, Marks 8**

10.5 RAM (Random Access Memories)

Dec.-03, 05, 06, 07, 08, 12, May-06, 09, 10, 12

There are two types of RAMs :

- Static RAM
- Dynamic RAM

10.5.1 Static RAM (SRAM)

Memories that consists of circuits capable of retaining their state as long as power is applied are known as **static memories**. These are random access memory (RAM) and hence combinely called **static RAM memories**.

10.5.1.1 Static RAM Cell

The Fig. 10.5.1 shows the implementation of static RAM cell. It consists of two cross-coupled inverters as a latch and two transistors T_1 and T_2 which act as a switches.

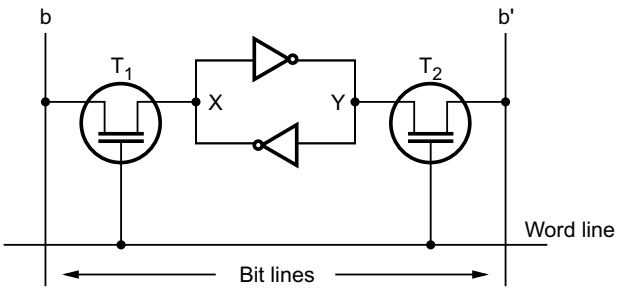


Fig. 10.5.1 Static RAM Cell

The latch is connected to two bit lines by transistors T_1 and T_2. The word line controls the opening and closing of transistors T_1 and T_2. When word line is at logic 0 level (Ground level), the transistors are off and the latch retains its state.

Read operation

For read operation, word line is made logic 1 (high) so that both transistors are ON. Now if the cell is in state 1, the signal on bit line b is high and the signal on bit line b′ is low. The opposite is true if the cell is in state 0. The b and b′ are complements of each other. The sense/write circuits connected to the bit lines monitor the states of b and b′ and set the output accordingly.

Write operation

For write operation, the state to be set is placed on the line b and its complement is placed on line b′ and then the word line is activated. This action forces the cell into the corresponding state and write operation is completed.

10.5.1.2 Bipolar RAM Cell

Fig. 10.5.2 shows a simplified schematic of a bipolar memory cell. The memory cell is implemented using TTL (Transistor - Transistor-Logic) multiple emitter technology. It stores 1-bit of information. It is nothing but a flip-flop. It can store either 0 or 1 as long as power is applied and it can set or reset to store either 1 or 0, respectively.

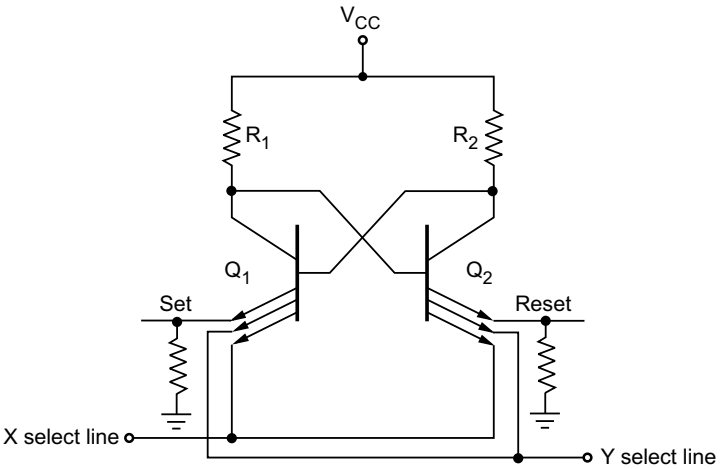


Fig. 10.5.2 TTL RAM cell

Operation : The X select and Y select input lines select a cell from matrix. The Q_1 and Q_2 are cross coupled inverters, hence one is always OFF while the other is ON. A "1" is stored in the cell if Q_1 is conducting and Q_2 is OFF. A "0" is stored in the cell if Q_2 is conducting and Q_1 is OFF. The state of the cell is changed to a "0" by pulsing a HIGH on the Q_1 (SET) emitter. This turns OFF Q_1. When Q_1 is turned OFF, Q_2 is turned ON. As long as Q_2 is ON, its collector is LOW and Q_1 is held OFF. A 1 can be rewritten by pulsing the Q_2(reset) emitter high.

10.5.1.3 MOSFET RAM Cell

Fig. 10.5.3 shows a simplified schematic of MOS static RAM cell. Enhancement mode MOSFET transistors are used to make this RAM cell. It is very similar to TTL cell discussed earlier.

Here, T_1 and T_2 form the basic cross coupled inverters and T_3 and T_4 act as load resistors for T_1 and T_2. X and Y lines are used for addressing the cell. When X and Y both are high, cell is selected. When X = 1, T_5 and T_6 get ON and the cell is connected to the data and $\overline{\text{data}}$ line. When Y = 1, T_7 and T_8 are made ON. Due to this, either read or write operation is possible.

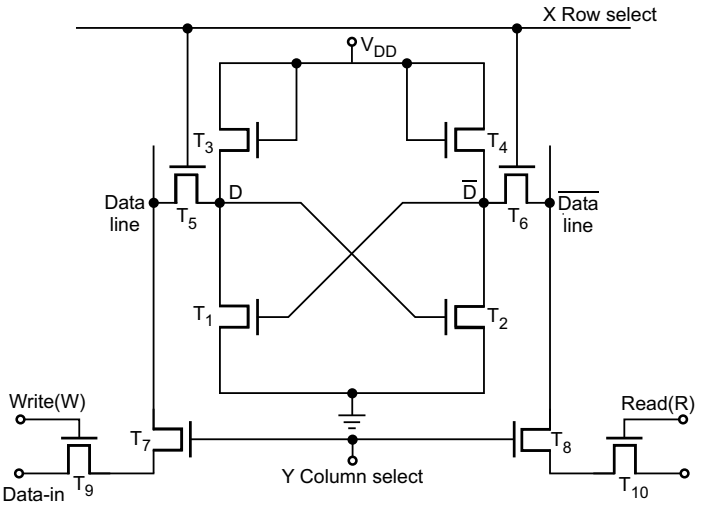


Fig. 10.5.3 MOS static RAM cell

Write operation : Write operation can be enabled by making W signal high. With write operation enabled, if data-in signal is logic 1, node D is also at logic 1. This turns ON T_2 and T_1 is cut-off. If new data on data-in pin is logic 0, T_2 will be cut-off and T_1 will be turned ON.

Read operation : Read operation can be enabled by making R signal high. With read operation enabled, T_{10} becomes ON. This connects the data output ($\overline{\text{Data}}$) line to the data out and thus the complement of the bit stored in the cell is available at the output.

10.5.2 Dynamic RAM (DRAM)

10.5.2.1 Dynamic RAM Cell

Dynamic RAM stores the data as a charge on the capacitor. Fig. 10.5.4 shows the dynamic RAM cell. A dynamic RAM contains thousands of such memory cells. When COLUMN (Sense) and ROW (Control) lines go high, the MOSFET conducts and charges the capacitor. When the COLUMN and ROW lines go low, the MOSFET opens and the capacitor retains its charge. In this way, it stores 1 bit. Since only a single MOSFET and capacitor are needed, the dynamic RAM contains more memory cells as compared to static RAM per unit area.

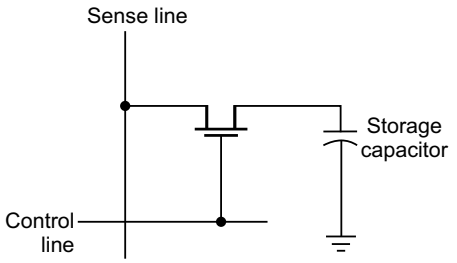


Fig. 10.5.4 Dynamic RAM

The disadvantage of dynamic RAM is that it needs refreshing of charge on the capacitor after every few milliseconds. This complicates the system design, since it requires the extra hardware to control refreshing of dynamic RAMs.

10.5.2.2 Comparison between SRAM and DRAM

Sr. No.	Static RAM	Dynamic RAM
1.	Static RAM contains less memory cells per unit area.	Dynamic RAM contains more memory cells as compared to static RAM per unit area.
2.	It has less access time hence faster memories.	Its access time is greater than static RAMs.
3.	Static RAM consists of number of flip-flops. Each flip-flop stores one bit.	Dynamic RAM stores the data as a charge on the capacitor. It consists of MOSFET and the capacitor for each cell.
4.	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors after every few milliseconds. Extra hardware is required to control refreshing. This makes system design complicated.
5.	Cost is more.	Cost is less.

Review Questions

1. *Draw a RAM cell and explain its working.* **Dec.-03, Marks 6**
2. *Draw the basic dynamic memory cell.* **Dec.-05, Marks 2**
3. *Write a note on dynamic RAM cell.* **Dec.-05, Marks 8**
4. *What is RAM ?* **Dec.-06, Marks 2**
5. *Draw a dynamic RAM cell and explain its operation. Compare its simplicity with that of NMOS static RAM cell, by way of diagram and operation.* **May-06, Marks 16**
6. *Draw the logic diagram of a memory cell.* **Dec.-07, Marks 2**
7. *Draw the block diagram of dynamic RAM cell.* **Dec.-08, Marks 2**
8. *Compare static RAMs and dynamic RAMs.* **May-09, 12, Dec.-09, Marks 8**
9. *What are the advantages of static RAM compared to dynamic RAM ?* **May-10, Marks 2**
10. *Explain the principle of operation of bipolar SRAM cell.* **May-10, Marks 8**
11. *Draw and explain the MOSFET RAM cell.*
12. *Draw the logic diagram of static RAM cell and bipolar RAM cell.* **Dec.-12, Marks 10**

10.6 RAM Organization

Dec.-05

RAM memory cells are organized in the form of an array, in which each cell is capable of storing one-bit of information. The Fig. 10.6.1 shows the row and the column organization of a 8192-bit memory chip.

The chip has 13 address lines. The first seven address lines are connected to the column decoder to indicate one of the 128 columns. The remaining 6 address lines are connected to the row decoder to indicate one of 64 rows. Where the decoded row and column

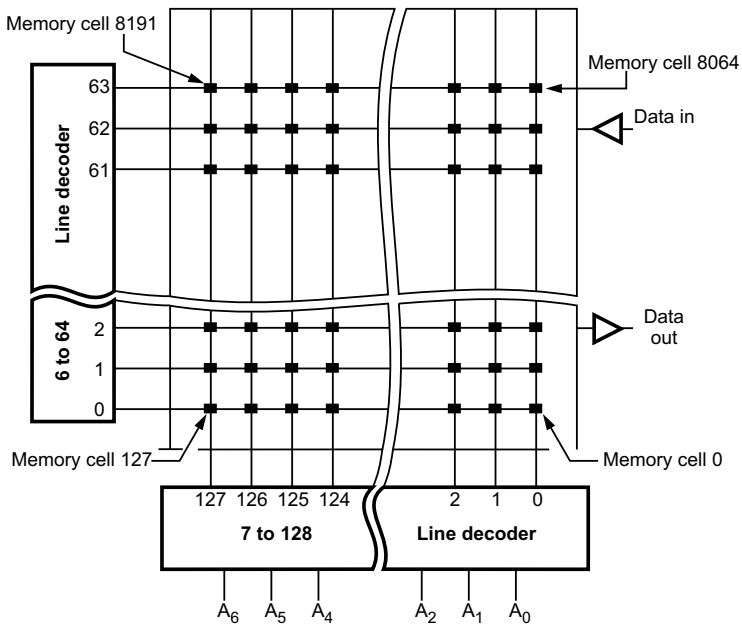


Fig. 10.6.1 Row and column organization for 8192-bit static RAM

cross, they select the desired individual memory cell. Simple arithmetic shows that there are $64 \times 128 = 8192$, crossings. Therefore, this memory has 8192 memory cells.

Review Question

1. *Describe the RAM organization.* **Dec.-05, Marks 12**

10.7 DRAM Organization

Dec.-11

The Fig. 10.7.1 shows the two dimensional address decoding scheme of typical DRAM. The DRAM shown in Fig. 10.7.1 is 16 M-bit DRAM. It is configured as $2M \times 8$.

The cells are organized in the form of a $4K \times 4K$ array. The 4096 cells in each row are addressed by 12 address bits and are divided in 512 groups of 8, so that a row can store 512×8, i.e. 512 bytes of data. Another 9 bits are used to specify a group of 8 bits in the selected row. Thus, a 21-bit address identifies a byte in this memory.

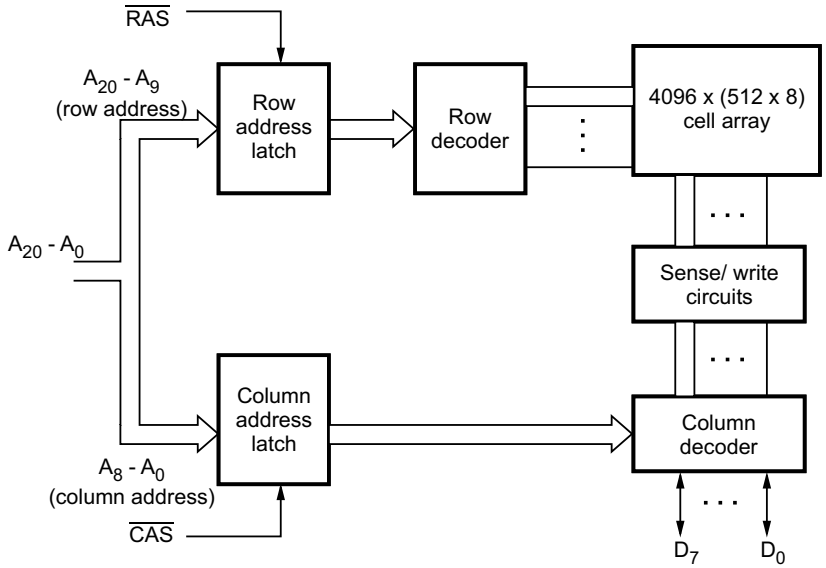


Fig. 10.7.1 (a) Internal organization of a $2M \times 8$ dynamic memory chip

Fig. 10.7.1 (b) shows another typical DRAM memory organisation. It is configured as $2M \times 4$. In DRAMs row address and column address lines are multiplexed to reduce number of pins. Thus for a given memory a DRAM chip has less address pins than SRAM chip. The elements of memory array are connected by both horizontal (row) and vertical (column) lines. Each horizontal line connects to the select terminal of each cell in its row; each vertical line connects to the Data-In/ sense terminal of each cell in its column.

The DRAM organisation shown in Fig. 10.7.1 (b) has four arrays of 2048 by 2048 elements. The 11 address lines are required to select one of 2048 rows. These 11 lines are fed into a row decoder, which has 11 lines of input and 2048 lines for output. The logic of the decoder activates a single one of the 2048 outputs depending on the bit pattern on the 11 input lines ($2^{11} = 2048$).

An additional 11 address lines select one of 2048 columns of four bits per column. Four data lines are used for the input and output of four bits to and from a data buffer.

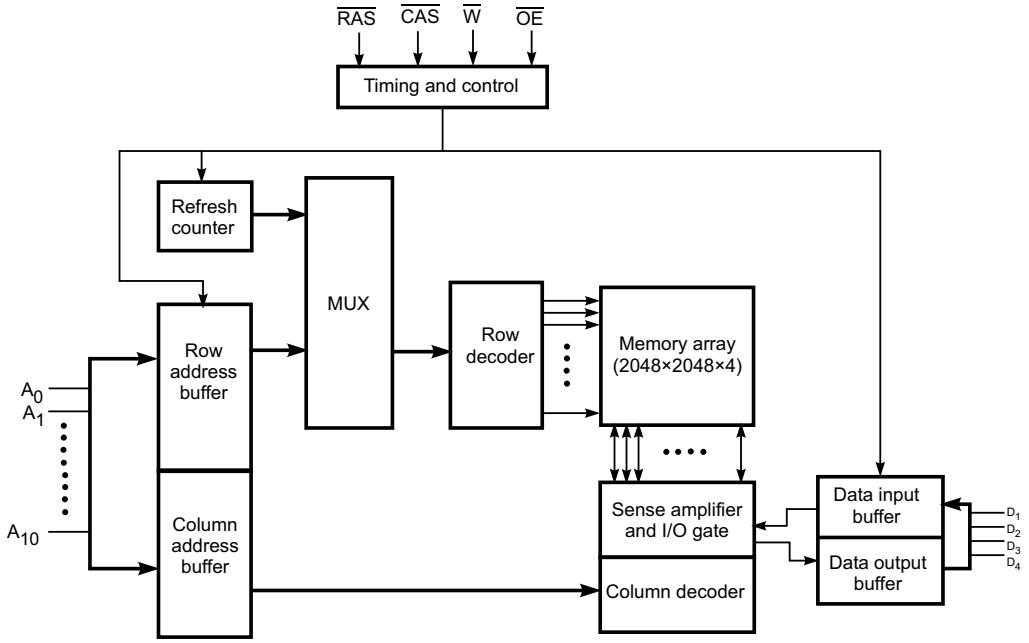


Fig. 10.7.1 (b) DRAM organization

Review Questions

1. *Explain the DRAM organization of 2M × 8 memory chip.*
2. *Describe the two dimensional address decoding scheme of typical DRAM in detail.*

Dec.-11, Marks 8

10.8 Memory Cycles and Timing Waveforms

May-08, 09, Dec.-08, 10

Let us study read and write memory cycle with their timing parameters.

Read Cycle

Fig. 10.8.1 shows the read cycle for static RAM. The timing diagram is drawn on the basis of different timing parameters.

These are as follows :

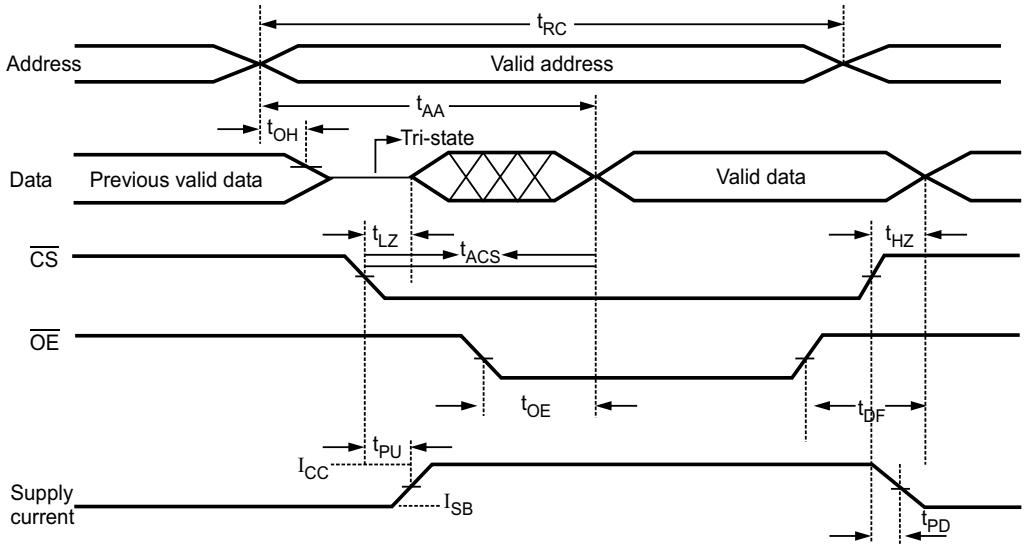


Fig. 10.8.1 Read cycle timing waveforms

1. t_{RC} : Read Cycle Time

It is the minimum time for which an address must be held stable on the address bus, in read cycle.

2. t_{AA} : Address Access Time

It is the maximum specified time within which a valid new data is put on the data bus after an address is applied.

3. t_{OH} : Output Hold from Address Change

It is the time for which a data from previously read address will be present on the output after a new address is applied.

4. t_{LZ} : Chip Selection to Output in Tri-state

It is the minimum time for which the data output will remain in tri-state after $\overline{CS}$ goes low.

5. t_{ACS} : Chip Select Access Time

It is the minimum time required for memory device to put valid data on the data bus after $\overline{CS}$ goes low.

6. t_{HZ} : Chip Deselection to Output in Tri-state

It is the time within which output returns to its high impedance state (tri-state) after $\overline{CS}$ goes high.

7. t_{OE} : Output Enable to Output Delay

It is the minimum time required for the memory device to put valid data on the data bus after $\overline{OE}$ goes low.

8. t_{DF} : Output Enable High to Output Float

It is the time for which data will remain valid on the data bus after $\overline{OE}$ goes high.

9. t_{PU} : Chip Selection to Power Up Time

It is the time within which the memory device is powered up to its normal operating current after $\overline{CS}$ goes low.

10. t_{PD} : Chip Deselection to Power Down Time

It is the time for which the memory device will remain powered up after $\overline{CS}$ goes high.

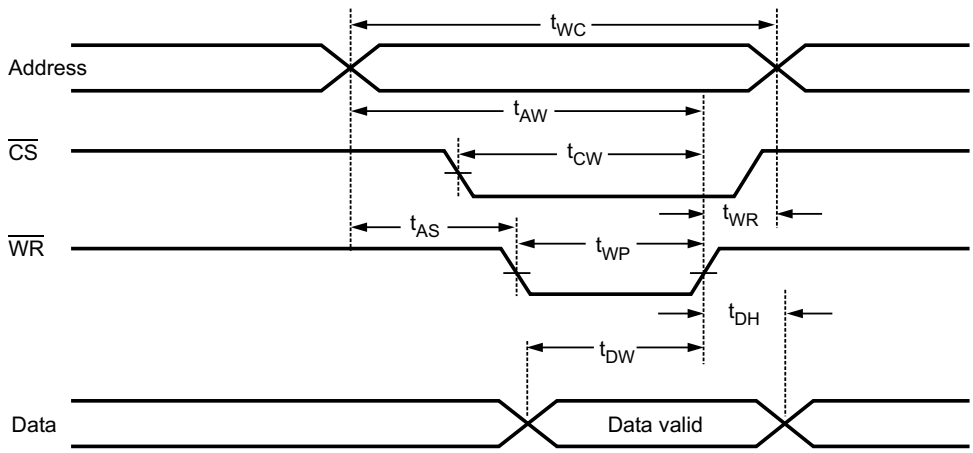


Fig. 10.8.2 Write cycle timing waveforms

Write Cycle

Fig. 10.8.2 shows the write cycle for static RAM. The timing diagram is drawn on the basis of different timing parameters. These are as follows :

1. t_{WC} : Write Cycle Time

It is the minimum time for which an address must be held stable on the address bus, in write cycle.

2. t_{AW} : Address Valid to End of Write

It is the time at which address must be applied on the address bus before $\overline{WR}$ goes high.

3. t_{WR} : Write Recovery Time

It is the time for which address will remain on address bus after $\overline{WR}$ goes high.

4. t_{AS} : Address Setup Time

When address is applied, it is the time after which $\overline{WR}$ can be made low.

5. t_{CW} : Chip Selection to the End of Write

It is the time at which the $\overline{CE}$ must be made low to select the device before $\overline{WR}$ goes high.

6. t_{WP} : Write Pulse Width

It is the time for which $\overline{WR}$ goes low.

7. t_{DW} : Data Valid to the End of Write

It is the minimum time for which data must be valid on the data bus before $\overline{WR}$ goes high.

8. t_{DH} : Data Hold Time

It is the time for which data must be held valid after $\overline{WR}$ goes high.

Review Questions

1. *What is write cycle time ?* **May-08, Marks 2**
2. *Explain 'write operation' with an example.* **Dec.-08, Marks 2**
3. *What is memory cycle ?* **May-09, Marks 2**
4. *Explain read and write operation of memory with timing waveforms.* **May-09, Dec.-10, Marks 8**
5. *What is access time and cycle time of a memory ?* **Dec.-10, May-09, Marks 8**

10.9 Memory Decoding

Dec.-03, 08, 12, May-07, 12

We have seen that in the organization of memory, there is a need for decoding circuit to select the memory word specified by the input address. In this section we study the operation of the decoder. As shown in the Fig. 10.9.1 there are 16 words of 8-bits each. A memory with 16 words needs four address lines. The four address inputs go through a 4 × 16 decoder to select one of the sixteen words. The decoder is enabled with a memory enable input. When the memory enable is 0, all outputs of the decoder are 0

and none of the memory words are selected. With the memory select at 1, one of the sixteen words is selected, dictated by the value in the four address lines. Once a word has been selected, the read/write input determines the operation. During write operation, the data available in the input lines are transferred into the eight memory cells of the selected word. The memory cells that are not selected are disabled and their previous binary values remain unchanged.

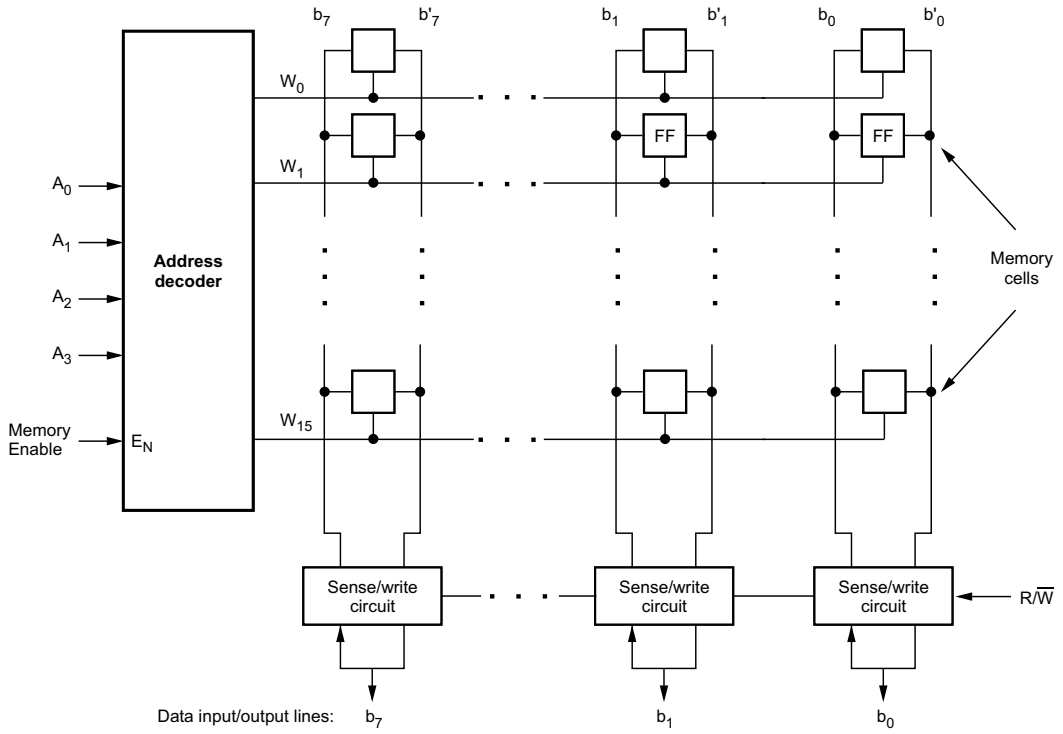


Fig. 10.9.1 Organization of bit cells in a memory chip

Coincident decoding

A decoder with k inputs and 2^k outputs requires 2^k AND gates with k inputs per gate. We can reduce the total number of gates and the number of inputs per gate by employing two decoders in a two dimensional selection scheme. The Fig. 10.9.2 shows the two dimensional decoding scheme. Here, two k/2 input decoders are used instead of one k-input decoder. One decoder performs the row selection and the other the column selection in a two dimensional matrix configuration.

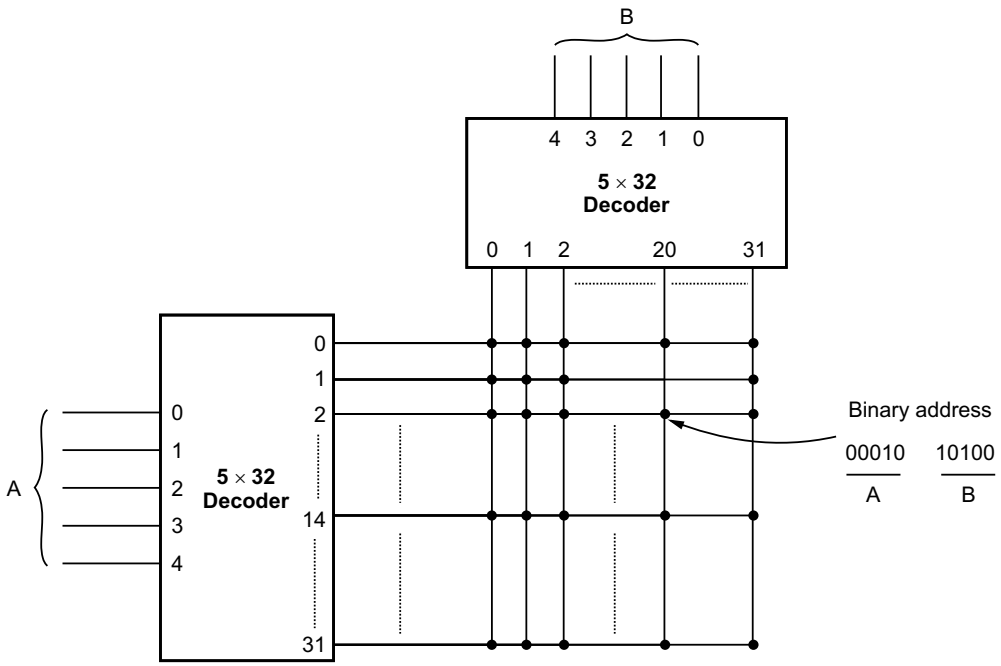


Fig. 10.9.2 Two dimensional decoding scheme

The two dimensional selection pattern shown in Fig. 10.9.2 has k = 10, i.e. 10 decoder inputs. These decoder inputs are divided by 2 get individual decoder inputs, i.e. 5 input for each decoder. Therefore, here instead of using a single 10 × 1024 decoder, we use two 5 × 32 decoders. With a single decoder we would need 1024 AND gates with 10 inputs in each. In the two-decoder case, we need 64 AND gates with five inputs in each. The five most significant bits of the address go to input A and the five least significant bits go to input B. Each word within the memory array is selected by the coincidence of one A and one B line. Thus, each word in memory is selected by the coincidence between 1 of 32 rows and 1 of 32 columns for a total of 1024 words.

Example 10.9.1 *Illustrate the concept of 16 × 8 bit ROM arrange with diagram.*

Solution :

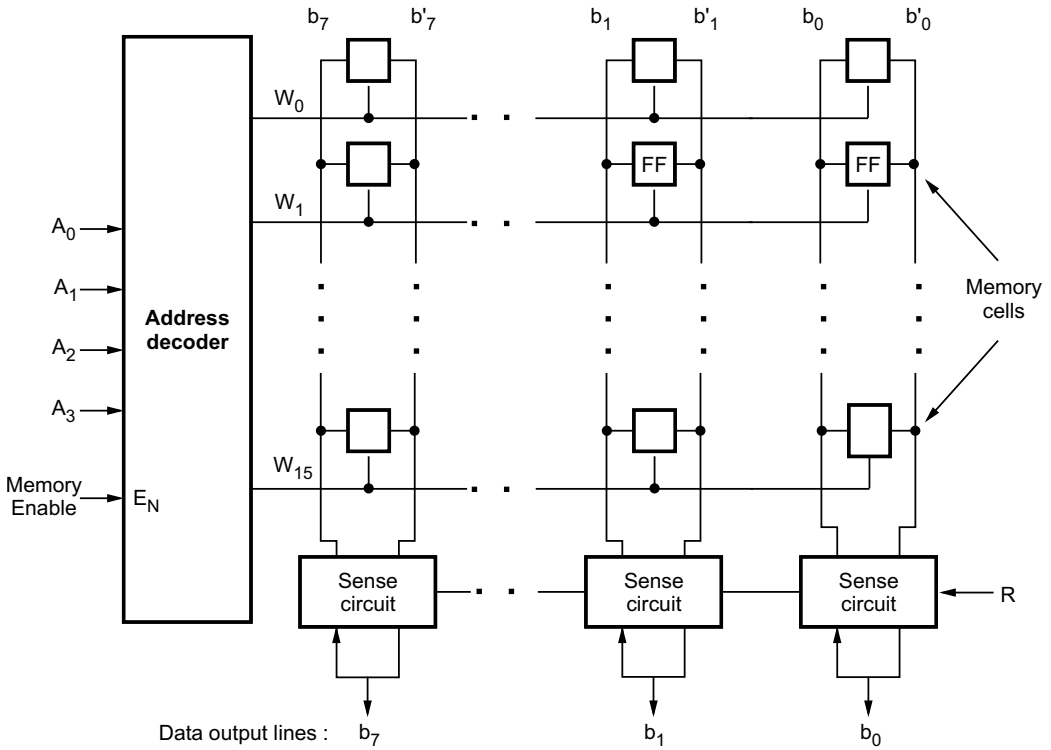


Fig. 10.9.3 Organization of bit cells in a memory chip

Example 10.9.2 *Explain the basic structure of a 256 × 4 static RAM, with neat diagram.*

Dec.-06, May-07, Marks 10

Solution : The Fig. 10.9.4 shows the basic structure of a 256 × 4 static RAM. Refer section 10.9 for explanation. Refer Fig. 10.9.4 on next page.

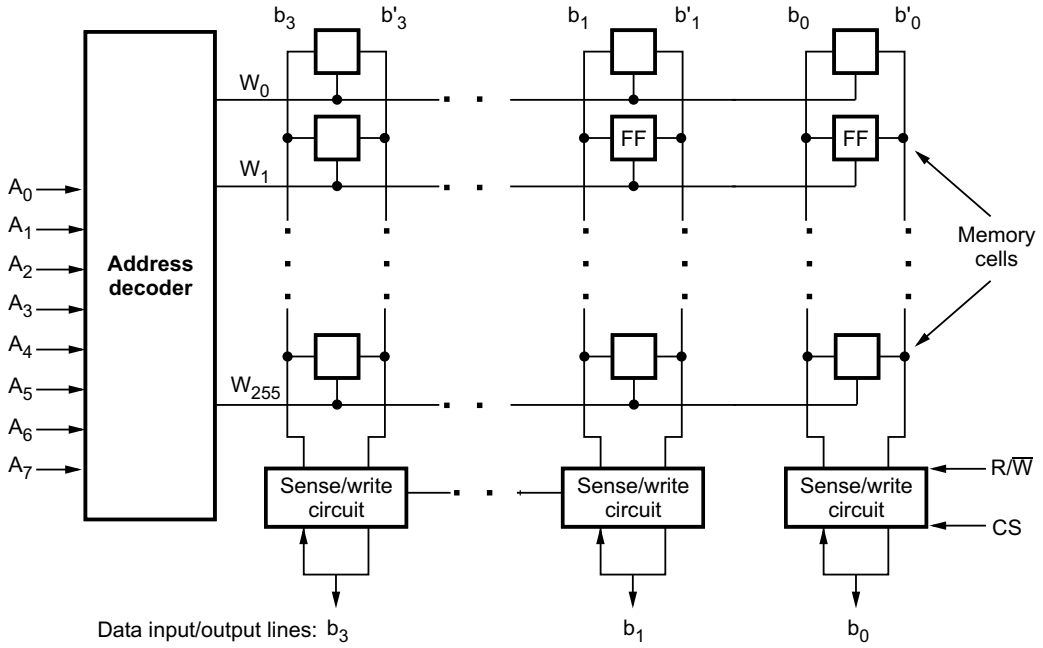


Fig. 10.9.4 Organization of bit cells in a memory chip

Review Question

1. *Explain memory decoding.* **Dec.-03, 08, 12, May-12, Marks 6**

10.10 Memory Expansion

May-10,12, Dec.-12

The memory expansion can be achieved in two ways : By expanding word size and by expanding memory capacity.

10.10.1 Expanding Word Size

It is possible to expand word size of memory by connecting two or more ICs together. The word size of memory IC can be increased by connecting two memory ICs in such a way that their data bus is in series and address bus in parallel. Both memory ICs are selected simultaneously by common chip select signal to access entire expanded word at a time. The example 10.10.1 illustrates method of word expansion. The expansion of word size is limited by the databus width.

Example 10.10.1 *Design 1 K × 8 RAM using two 1 K × 4 ICs.*

Solution :

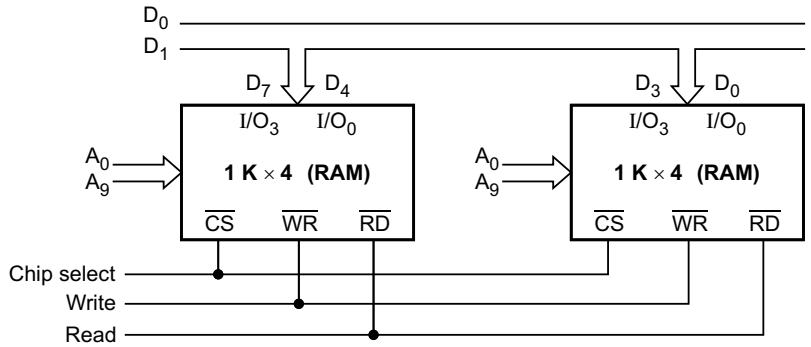


Fig. 10.10.1

10.10.2 Expanding Memory Capacity

The memory capacity can be increased by connecting two or more memory ICs in parallel. This means that the address, data and control lines are connected in parallel to all memory ICs. Each IC is selected by the separate chip select signal generated by the address decoder. This is illustrated by the following example. The expansion of memory capacity is limited by address bus width.

Example 10.10.2 *Design 16 K × 8 RAM using four 4 K × 8 ICs.*

Solution :

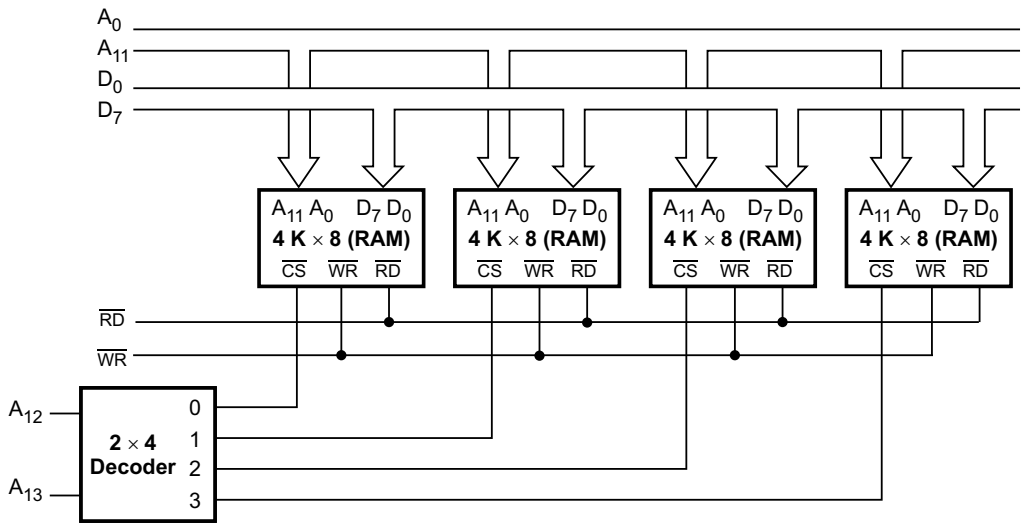


Fig. 10.10.2

10.10.3 Limitations for Memory Expansion

The memory devices are interfaced with processors and are accessed using address, data and control bus provided by the processor. Each processor has defined number of address lines and data lines. Suppose a processor has 24 address lines and 16 data lines, we can expand memory word size up to 16 and we can expand memory capacity upto 2^{24} = 16 Mbytes.

Illustrative Examples

Example 10.10.3 *How can one make 64 × 8 ROM using four 32 × 4 ROMs ? Draw such a circuit and explain.* **May-06, Marks 8**

Solution : The Fig. 10.10.3 shows the 64 × 8 ROM using four 32 × 4 ROMs. Here, two pairs ICs are connected such that their data bus is in series and address bus is in parallel. The pairs are connected in parallel so that their address, data and control lines are in parallel. To address 32 memory locations in each memory we need 5 address lines. The additional address line is used to select the particular pair.

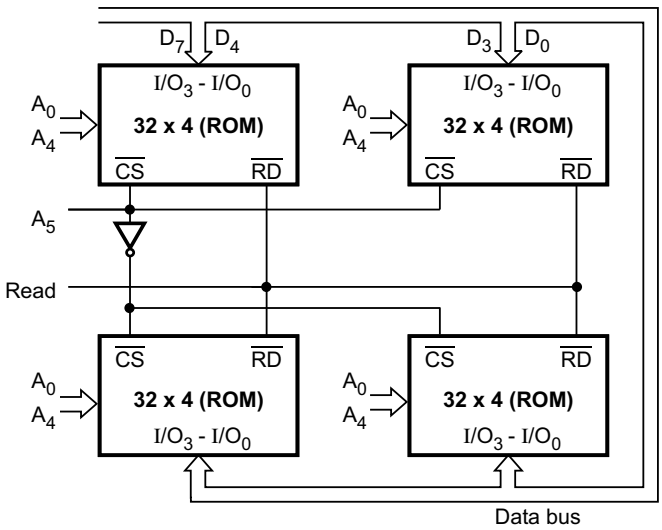


Fig. 10.10.3

Example 10.10.4 *Given the 32 × 8 ROM chip with enable input, show the external connection necessary to construct a 128 × 8 ROM with four chips and a decoder.* **Dec.-11, Marks 8**

Solution :

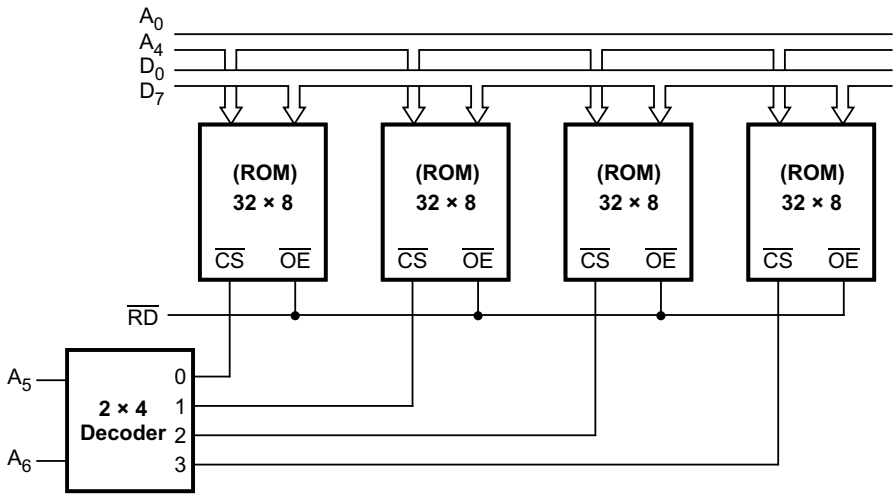


Fig. 10.10.4

Review Question

1. *What is meant by memory expansion? Mention its limit.* **May-10, 12, Dec.-12, Marks 2**

10.11 Programmable Logic Devices (PLDs)

May-03, 05, 07, 09, 11, Dec.-03, 06, 09, 12

So far we have discussed various digital ICs for performing basic digital operations and other functions, such as adders, comparators, arithmetic logic unit, multiplexers, demultiplexers, code converters, shift registers, counters etc. These ICs, due to their fix function are known as **fixed function ICs.** These ICs are designed by their manufacturers and produced in large quantities to satisfy the needs of a wide variety of applications.

We have seen the design of digital circuits using fixed function ICs. There are two more approaches for the design of digital circuits.

- Use of Application Specific Integrated Circuits (ASICs)
- Use of Programmable Logic Devices (PLDs)

In the fixed function IC approach, we have to use various fixed function ICs to implement different functional blocks in the digital circuit. On the other hand, in ASIC, a single IC is designed and manufactured to implement the entire circuit. In the third approach programmable logic devices are used to implement logic functions. The main advantage of PLD approach is that PLDs can be easily configurable by the individual user for specific applications. The Table 10.11.1 shows the comparison between these three design approaches.

Comparison parameter	Fixed-function IC approach	ASIC approach	PLD approach
Development cost	Low	High	Low
Space required	Large	Minimum	Less
Power required	Large	Less	Less
Reliability	Less compared to other two approaches.	Highest	High
Circuit testing	Easy	Specialized testing methods are required with may increase cost and effort.	Easy
Design flexibility	Less	No	More
Modification in design	Possible with change in circuit and/or with change in components.	No	May be possible without any circuit or component changes but only by reconfigurating the device.
Design security	Lack of security i.e. circuit can easily be copied by others.	High	High
Design time	Less	More	Less

Table 10.11.1 Comparison between design approaches

PLDs can be reprogrammed in few seconds and hence gives more flexibility to experiment with designs. Reprogramming feature of PLDs also make it possible accept changes/modifications in the previously design circuits. These two main advantages and others discussed in Table 10.11.1 make PLDs very popular in digital design.

According to architecture, complexity and flexibility in programming PLDs are classified as

- PROMs : Programmable Read Only Memories
- PLAs : Programmable Logic Arrays
- PAL : Programmable Array Logic
- FPGAs : Field Programmable Gate Arrays
- CPLDs : Complex Programmable Logic Devices

10.11.1 PROM (Programmable Read Only Memory)

The Fig. 10.11.1 shows the block diagram of PROM. It consists of n-input lines and m-output lines. Each bit combination of the input variables is called an **address**. Each bit combination that comes out of the output

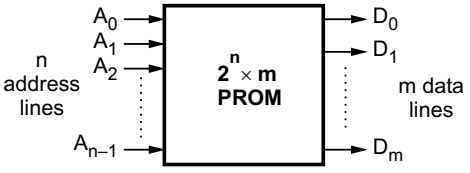


Fig. 10.11.1

lines is called a **word**. The number of bits per word is equal to the number of output lines, m. The address specified in binary number denotes one of the minterms of n variables. The number of distinct addresses possible with n-input variables is 2^n. An output word can be selected by a unique address and since there are 2^n distinct addresses in PROM, there are 2^n distinct words in the PROM. The word available on the output lines at any given time depends on the address value applied to the input lines.

Let us consider 64 × 4 PROM. The PROM consists of 64 words of 4-bits each. This means that there are four output lines and particular word from 64 words presently available on the output lines is determined from the six input lines. There are only six inputs in a 64 × 4 PROM because 2^6 = 64 and with six variables, we can specify 64 addresses or minterms. For each address input, there is a unique selected word. Thus, if the input address is 000000, word number 0 is selected and applied to the output lines. If the input address is 111111, word number 63 is selected and applied to the output lines.

The Fig. 10.11.2 shows the internal logic construction of a 64 × 4 PROM. The six input variables are decoded in 64 lines by means of 64 AND gates and 6 inverters. Each output of the decoder represents one of the minterms of a function of six variables. The 64 outputs of the decoder are connected through fuses to each OR gate. Only four of these fuses are shown in the diagram, but actually each OR gate has 64 inputs and each input goes through a fuse that can be blown as desired.

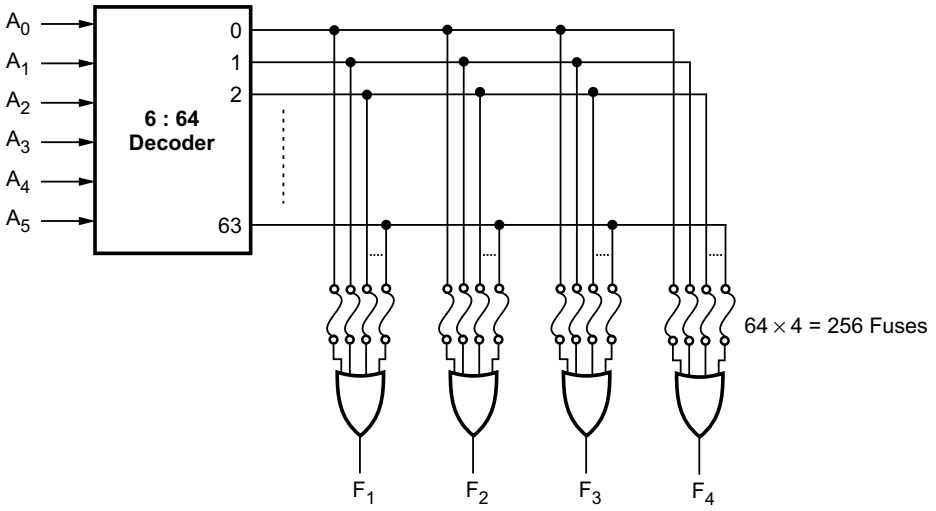


Fig. 10.11.2 Logic construction of 64 × 4 PROM

The PROM is a two level implementation in sum of minterms form. Let us see AND-OR and AND-OR-INVERTER implementation of PROM. Fig. 10.11.3 shows the 4 × 2 PROM with AND-OR and AND-OR-INVERTER implementations.

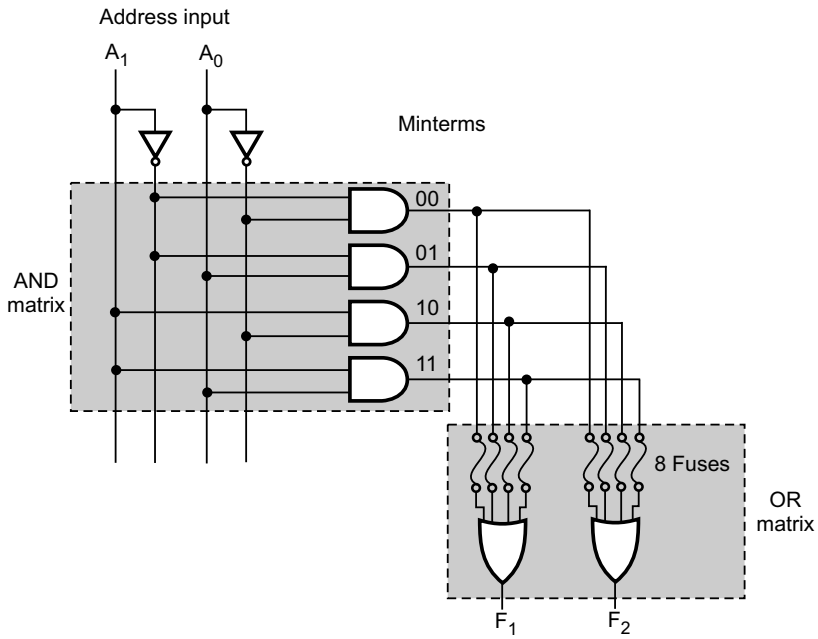


Fig. 10.11.3 (a) 4 × 2 PROM with AND-OR gates

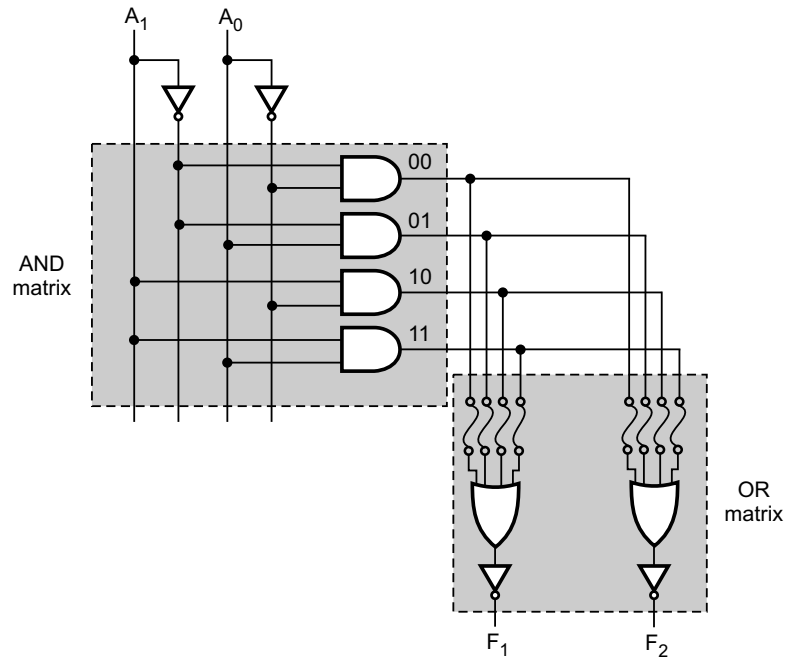


Fig. 10.11.3 (b) 4 × 2 PROM with AND-OR-INVERTER gates

10.11.1.1 AND Matrix

The Fig. 10.11.4 shows the AND matrix. It is used to form product terms. It has m AND gates with 2n-inputs and m-outputs, one for each AND gate. The Fig. 10.6.4 shows the AND gates formed by diodes and resistors structure. Each AND gate has all the input variables in complemented and uncomplemented form. There is a nichrome fuse link in series with each diode which can be burn out to disconnect particular input for that AND gate. Before programming, all fuse links are intact and the product term for each AND gate is given by

$$P = I_0 \cdot \overline{I}_0 \cdot I_1 \cdot \overline{I}_1 \ \ldots\ldots\ I_{n-1} \cdot \overline{I}_{n-1}$$

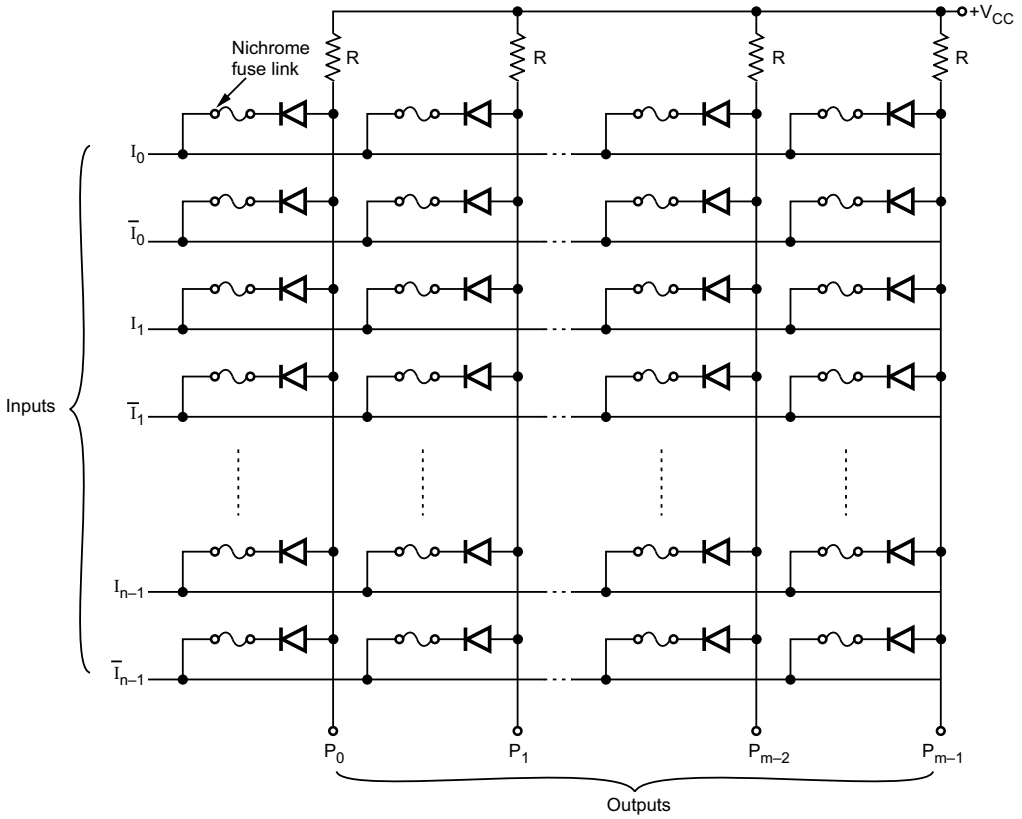


Fig. 10.11.4 Internal structure of AND matrix

The Fig. 10.11.5 shows the simplified and equivalent representation of input connections for one AND gate. The array logic symbol shown in Fig. 10.11.5 (b) uses a single horizontal line connected to the gate input and multiple vertical lines to indicate the individual inputs. Each intersection between horizontal line and vertical line indicates the fuse connection.

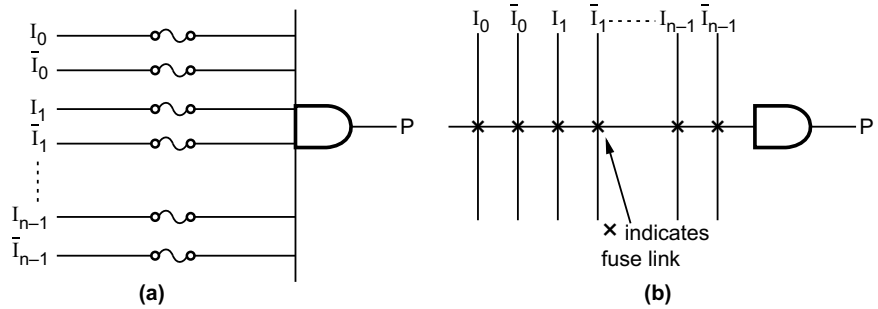


Fig. 10.11.5 Equivalent representation of AND gate

The Fig. 10.11.6 shows the simplified representation of AND matrix with input buffer.

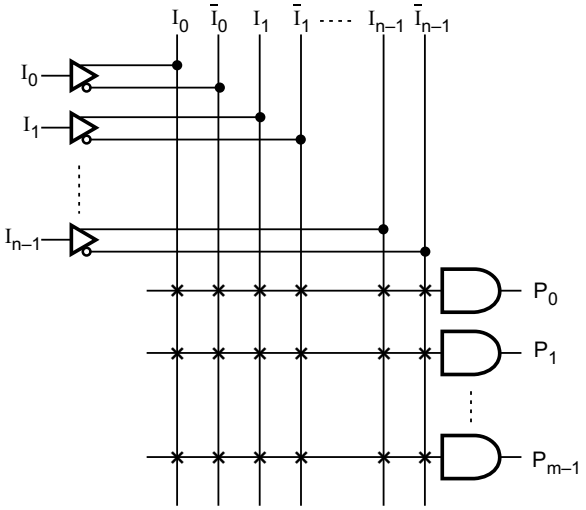


Fig. 10.11.6 Simplified representation of AND matrix with input buffer

10.11.1.2 OR Matrix

The OR matrix is provided to produce the logical sum of the product term outputs of the AND matrix. The Fig. 10.11.7 shows the OR gates formed by diodes and resistors structure. Each OR gate has all the product terms as input variables. There is a nichrome fuse link in series with each diode which can be burn out to disconnect particular product term for that OR gate. Before programming, all fuse link in OR matrix are also intact and the sum term for each OR gate is given by,

$$S = P_0 + P_1 + ... + P_{m-2} + P_{m-1}$$

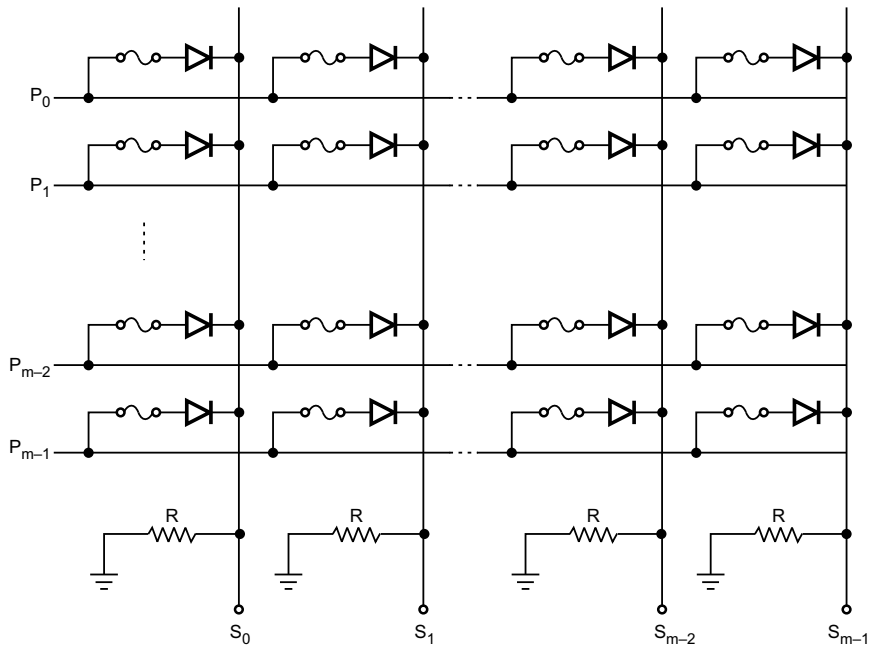


Fig. 10.11.7

The Fig. 10.11.8 shows the simplified and equivalent representation of input connections for one OR gate.

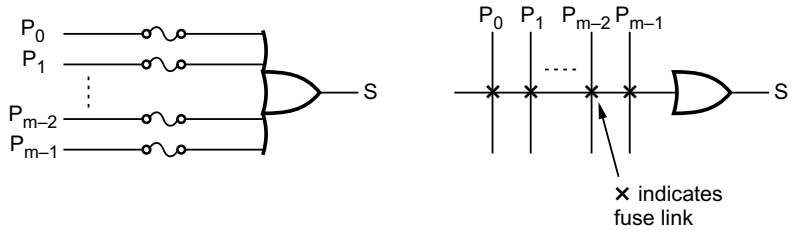


Fig. 10.11.8 Equivalent representation of OR gate

The Fig. 10.11.9 shows the simplified representation of OR matrix.

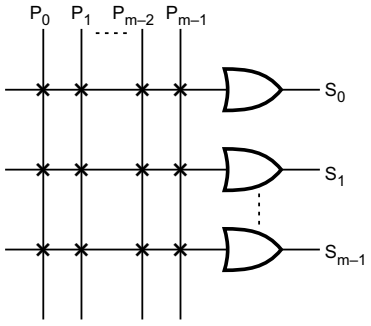


Fig. 10.11.9 Simplified representation of OR matrix

10.11.1.3 Invert / Non-invert Matrix

Invert/Non-invert matrix provides output in the complement or uncomplemented form. The user can program the output in either complement or uncomplement form as per design requirements. The typical circuits for invert/non-invert matrix is as shown in Fig. 10.11.10. In both the cases if fuse is intact the output is in its uncomplemented form; otherwise output is in the complemented form.

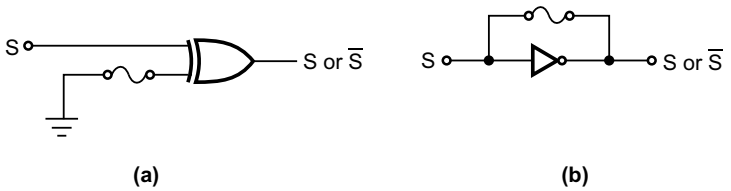


Fig. 10.11.10 Inverting and non-inverting circuits

10.11.1.4 Combinational Logic Implementation using PROM

Looking at the logic diagram of the PROM, we can realize that each output provides the sum of all the minterms of n-input variables. We know that any Boolean function can be expressed in sum of minterms form. By breaking the links of those minterms not included in the function, each PROM output can be made to represent the Boolean function of one of the output variables in the combinational circuit. For an n-input, m-output combinational circuit, we need a $2^n \times m$ PROM.

Examples for Understanding

Example 10.11.1 *Using PROM realize the following expressions.*

$$F_1(a, b, c) = \sum m(0, 1, 3, 5, 7)$$

$$F_2(a, b, c) = \sum m(1, 2, 5, 6)$$

Solution : The given functions have three inputs. They generate $2^3 = 8$ minterms and since there are two functions, there are two outputs. The functions can be realized as shown in Fig. 10.11.11.

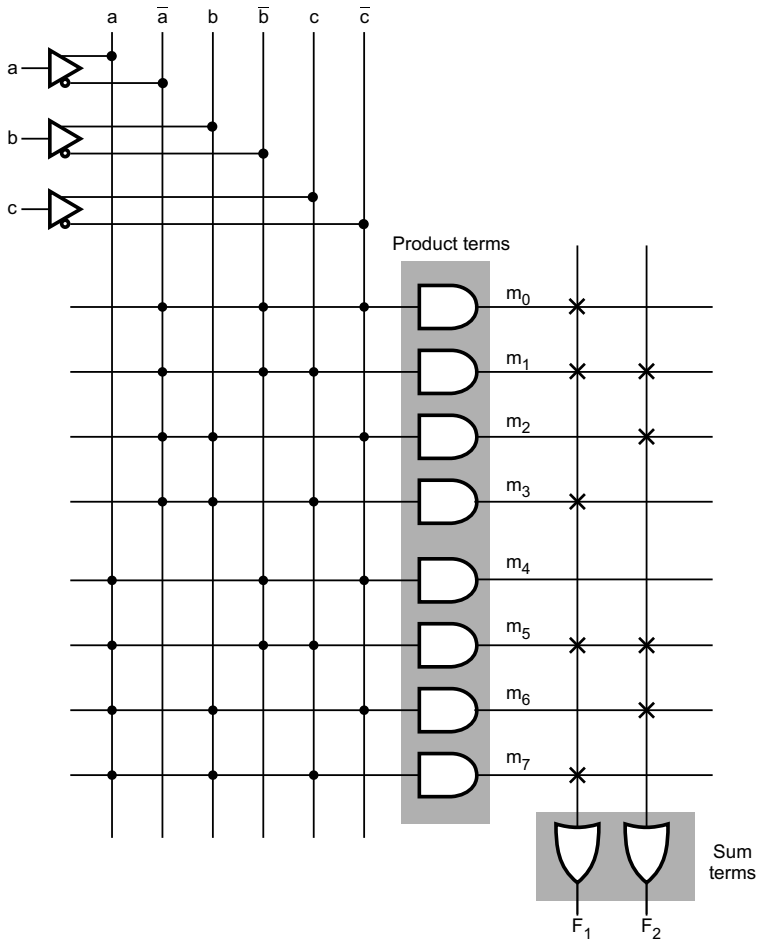
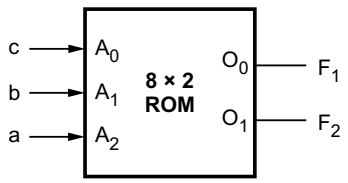


Fig. 10.11.11

The Fig. 10.11.12 shows the block diagram and truth table of PROM.



(a) Block diagram

A_2	A_1	A_0	F_1	F_2
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	1	0

(b) PROM truth table

Fig. 10.11.12

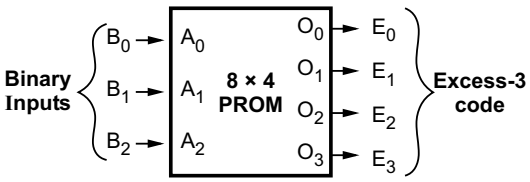
Example 10.11.2 *Design a combinational using a PROM. The circuit accepts 3-bit binary number and generates its equivalent Excess-3 code.*

Solution : Let us derive the truth table for the given combination circuit. Table 10.11.2 shows the truth table.

Inputs			Outputs			
B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

Table 10.11.2 Truth table for 3-bit binary to excess-3 converter

In practice when we are designing combinational circuits with PROM, it is not necessary to show the internal gate connections of fuses inside the unit, as shown in the Fig. 10.11.13. (Refer Fig. 10.11.13 on next page). This was shown for demonstration purpose only. The designer has to only specify the PROM (inputs and outputs) and its truth table, as shown in the Fig. 10.11.14.



(a) Block diagram

A_2	A_1	A_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0

(b) PROM truth table

Fig. 10.11.14

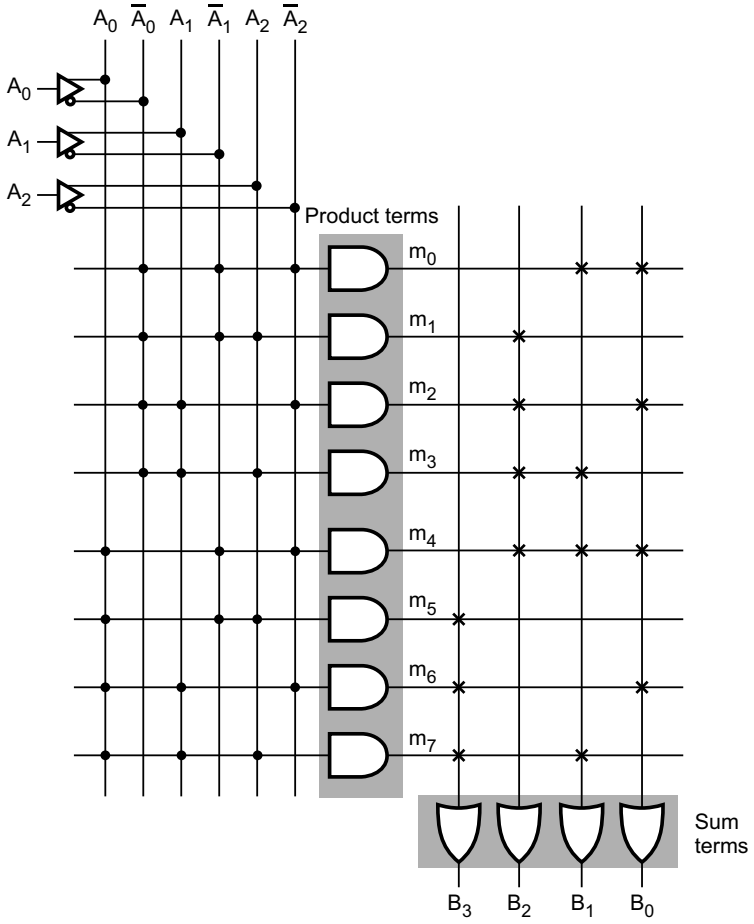


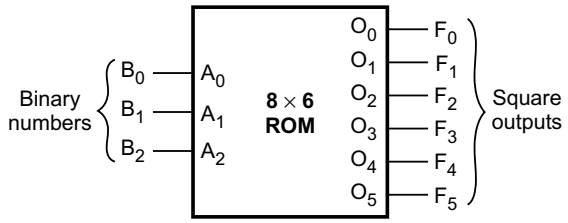
Fig. 10.11.13

Examples with Solutions

Example 10.11.3 *Design a combinational circuit using ROM. The circuit accepts 3-bit number and generates an output binary number equal to square of input number.*

Dec.-09, May-10, 11, Marks 16

Solution :



(a) Block diagram

Binary input on address lines			Square of number on data lines					
B_2	B_1	B_0	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

(b) ROM truth table

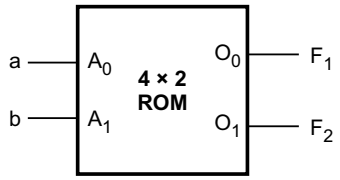
Fig. 10.11.15

Example 10.11.4 *Design ROM for the following functions*

$F_1 = \sum(1, 2, 3)$; $F_2 = \sum(0, 2)$

May-11, Marks 7

Solution : The given functions have 4 minterms. To generate four minterms and two outputs we need 4 × 2 ROM. For 4 × 2 ROM, there are two address inputs and two data outputs.



(a) Block diagram

Address inputs		Data outputs	
a	b	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1

(b) ROM truth table

Fig. 10.11.16

Example 10.11.5 *Implement binary to excess 3 code converter using ROM.*

May-06, Marks 8

Solution :

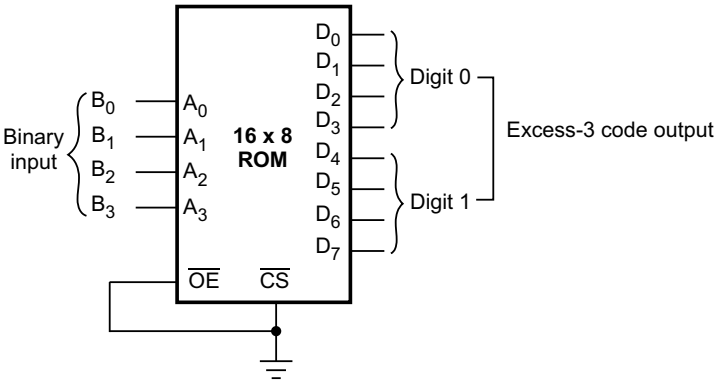


Fig. 10.11.17

Address				Memory contents							
A_3	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	0	1	1	0	1	0	0
0	0	1	0	0	0	1	1	0	1	0	1
0	0	1	1	0	0	1	1	0	1	1	0
0	1	0	0	0	0	1	1	0	1	1	1
0	1	0	1	0	0	1	1	1	0	0	0
0	1	1	0	0	0	1	1	1	0	0	1
0	1	1	1	0	0	1	1	1	0	1	0
1	0	0	0	0	0	1	1	1	0	1	1
1	0	0	1	0	0	1	1	1	1	0	0
1	0	1	0	0	1	0	0	0	0	1	1
1	0	1	1	0	1	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	1	0	1
1	1	0	1	0	1	0	0	0	1	1	0
1	1	1	0	0	1	0	0	0	1	1	1
1	1	1	1	0	1	0	0	1	0	0	0

Table 10.11.3 ROM contents

Example 10.11.6 *Implement a 2-bit multiplier using ROM.* **Dec.-10, Marks 2**

Solution :

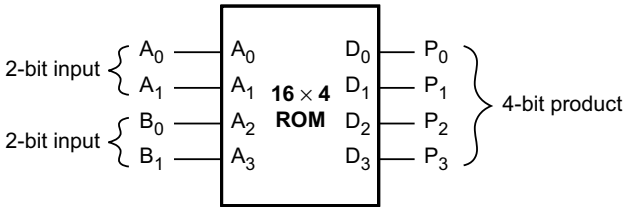


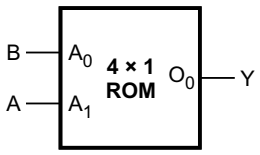
Fig. 10.11.18

ROM Truth Table

B_1	B_0	A_1	A_0	P_3	P_2	P_1	P_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Example 10.11.7 *Implement the exclusive-or function using ROM.* **May-11, Marks 2**

Solution : Implement



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

(a) Block diagram **(b) ROM truth table**

Fig. 10.11.19

Examples for Practice

Example 10.11.8 *Design a switching circuit that converts a 4 bit binary code into a 4 bit Gray code using ROM array.*

Example 10.11.9 *Design a 3-bit gray to binary code converter using suitable ROM.*

Example 10.11.10 *Implement binary to Gray code converter using PROM devices.*

Dec.-12, Marks 8

10.11.2 PLA (Programmable Logic Array)

The combinational circuit do not use all the minterms every time. Occasionally, they have don't care conditions. Don't care condition when implemented with a PROM becomes an address input that will never occur. The result is that not all the bit patterns available in the PROM are used, which may be considered a waste of available equipment.

For cases where the number of don't care conditions is excessive, it is more economical to use a second type of LSI component called a **Programmable Logic Array (PLA)**. A PLA is similar to a PROM in concept; however it does not provide full decoding of the variables and does not generates all the minterms as in the PROM. The PLA replaces decoder by group of AND gates, each of which can be programmed to generate a product term of the input variables. In PLA, both AND and OR gates have fuses at the inputs, therefore in PLA both AND and OR gates are programmable. Fig. 10.11.20 shows the block diagram of PLA. It consists of n-inputs, output buffer with m outputs, m product terms, m sum terms, input and output buffers. The product terms constitute a group of m AND gates and the sum terms constitute a group of m OR gates, called OR matrix. Fuses are inserted between all n-inputs and their complement values to each of the AND gates. Fuses are also provided between the outputs of the AND gates and the inputs of the OR gates. The third set of fuses in the output inverters allows the output function to be generated either in the AND-OR form or in the AND-OR-INVERT form. When inverter is bypassed by link we get AND-OR implementation. To get AND-OR-INVERTER implementation inverter link has to be disconnected.

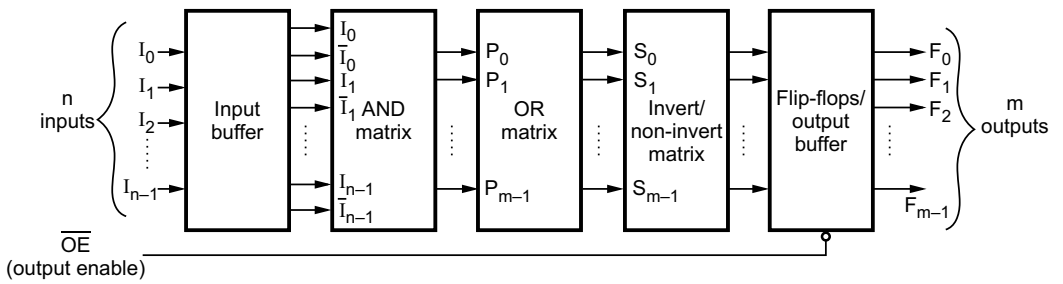


Fig. 10.11.20 Block diagram of a PLA

10.11.2.1 Input Buffer

Input buffers are provided in the PLA to limit loading of the sources that drive the inputs. They also provide inverted and non-inverted form of inputs at its output. The Fig. 10.11.21 shows two ways of representing input buffer for single input.

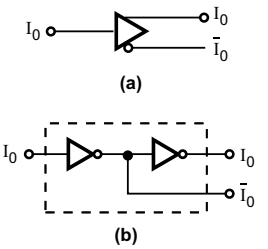


Fig. 10.11.21 Input buffer for single input line

10.11.2.2 Output Buffer

The driving capacity of PLA is increased by providing buffers at the output. They are usually TTL compatible. The Fig. 10.11.22 shows the tri-state, TTL compatible output buffer. The output buffer may provide totem-pole, open collector or tri-state output.

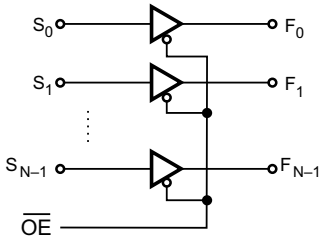


Fig. 10.11.22 Output buffers

10.11.2.3 Output through Flip-Flops

For the implementation of sequential circuits we need memory elements, flip-flops and combinational circuitry for deriving the flip-flop inputs. To satisfy both the needs some PLAs are provided with flip-flop at each output, as shown in the Fig. 10.11.23.

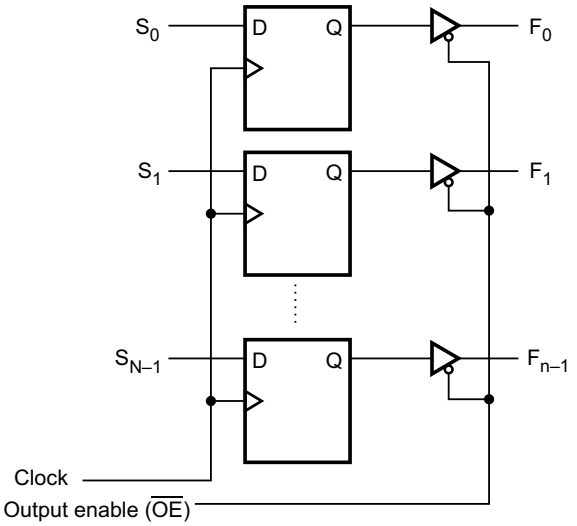


Fig. 10.11.23 PLA with flip-flop at the output

10.11.2.4 Implementation of Combination Logic Circuit using PLA

Like ROM, PLA can be **mask-programmable** or **field-programmable**. With a mask-programmable PLA, the user must submit a PLA program table to the manufacturer. This table is used by the vendor to produce a user-made PLA that has the required internal paths between inputs and outputs. A second type of PLA available is

called a **field-programmable logic array** or FPLA. The FPLA can be programmed by the user by means of certain recommended procedures. FPLAs can be programmed with commercially available programmer units.

As mentioned earlier, user has to submit PLA program table to the manufacturers to get the user-made PLA. Let us study how to determine PLA program table with the help of example.

Examples for Understanding

Example 10.11.10 *A combinational circuit is defined by the functions :*

$F_1 = \sum m\ (3, 5, 7)$, $F_2 = \sum m\ (4, 5, 7)$

Implement the circuit with a PLA having 3 inputs, 3 product terms and two outputs.

Solution :

Step 1 : Simplify the given Boolean functions

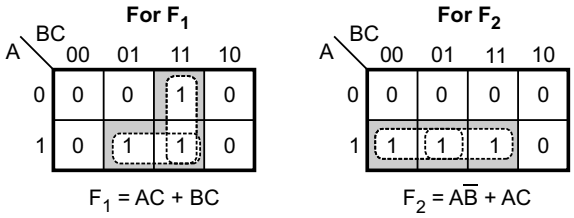


Fig. 10.11.24

The Boolean functions are simplified, as shown in the Fig. 10.11.21. The simplified functions in sum of products are obtained from the maps are :

$$F_1 = AC + BC, \quad F_2 = A\overline{B} + AC$$

Step 2 : Write PLA program table

	Product term	Inputs			Outputs	
		A	B	C	F_1	F_2
AC	1	1	–	1	1	1
BC	2	–	1	1	1	–
$A\overline{B}$	3	1	0	–	–	1
					T	T	T/C

Table 10.11.4 PLA program table

Therefore, there are three distinct product terms : AC, BC and $A\overline{B}$, and two sum terms. The PLA program table shown in Table 10.11.4 consists of three columns specifying product terms, inputs and outputs. The first column gives the lists of product terms numerically. The second column specifies the required paths between inputs and

AND gates. The third column specifies the required paths between the AND gates and the OR gates. Under each output variable, we write a T (for true) if the output inverter is to be bypassed, and C (for complement) if the function is to be complemented with the output inverter. The product terms listed on the left of first column are not the part of PLA program table they are included for reference only.

Step 3 : Implementation

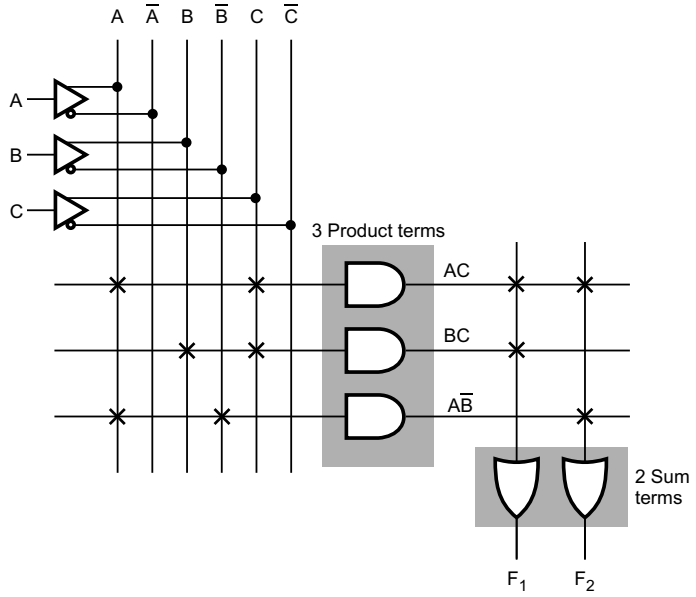


Fig. 10.11.25

Example 10.11.11 *Draw a PLA circuit to implement the logic functions* $\overline{A}BC + A\overline{B}C + A\overline{C}$ *and* $\overline{A}\,\overline{B}\,\overline{C} + BC$. **Dec.-08, Marks 6**

Solution :

Step 1 : Simplify the Boolean functions

$$\overline{A}\,B\,C + A\,\overline{B}\,C + A\overline{C} = \overline{A}\,B\,C + A\,(\overline{B}\,C + \overline{C})$$

$$= \overline{A}\,B\,C + A\,(\overline{B} + \overline{C}) \qquad \because A + \overline{A}\,B = A + B$$

$$= \overline{A}\,B\,C + A\,\overline{B} + A\overline{C}$$

Note : The second Boolean function is in simplified form.

Step 2 : Implementation

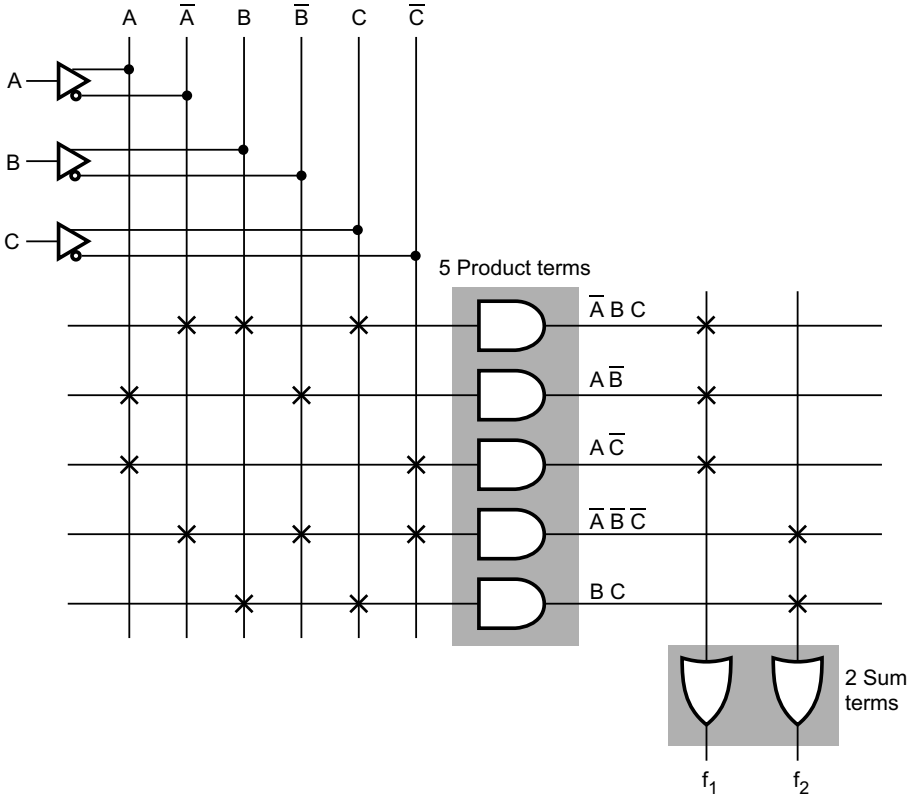


Fig. 10.11.26

Example 10.11.12 *Implement the following multiboolean function using* $3 \times 4 \times 2$ *PLA PLD.*

$f_1(a_2, a_1, a_0) = \sum m(0, 1, 3, 5)$ *and*

$f_2(a_2, a_1, a_0) = \sum m(3, 5, 7)$

Solution :

Step 1 : Simplify the Boolean functions.

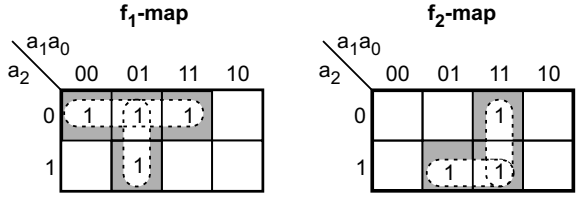


Fig. 10.11.27 K-map simplification

$$\therefore \quad f_1 = \bar{a}_2\,\bar{a}_1 + \bar{a}_2\,a_0 + \bar{a}_1\,a_0$$

$$f_2 = a_2\,a_0 + a_1\,a_0$$

To implement functions f_1 and f_2 we require $3 \times 5 \times 2$ PLA and we have to implement them using $3 \times 4 \times 2$ PLA. Therefore, we have to examine product terms by grouping 0s instead of 1. That is product terms for complement of a function.

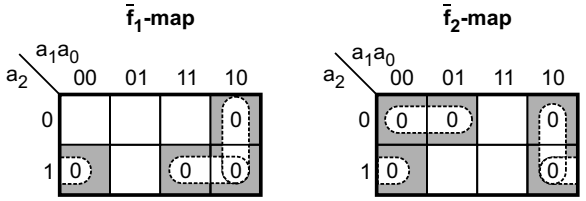


Fig. 10.11.28

$$\therefore \quad \bar{f}_1 = a_2\,\bar{a}_0 + a_1\,\bar{a}_0 + a_2\,a_1$$

$$\bar{f}_2 = \bar{a}_2\,\bar{a}_1 + a_1\,\bar{a}_0 + a_2\,\bar{a}_0$$

Step 2 : Implementation

Looking at function outputs we can realize that product terms $a_2\,\bar{a}_0$ and $a_1\,\bar{a}_0$ are common in both functions. Therefore, we need only 4 product terms and functions can be implemented using a $3 \times 4 \times 2$ PLA as shown in Table 10.11.9 and Fig. 10.11.29.

PLA

Product terms	Inputs			Outputs	
	a_2	a_1	a_0	f_1	f_2
$a_2\,\bar{a}_0$	1	–	0	1	1
$a_1\bar{a}_0$	–	1	0	1	1
$a_2\,a_1$	1	1	–	1	–
$\bar{a}_2\bar{a}_1$	0	0	–	–	1
				C	C

Table 10.11.5

As shown in the Fig. 10.11.29 exclusive-OR gate is programmed to invert the function to get the desired function outputs.

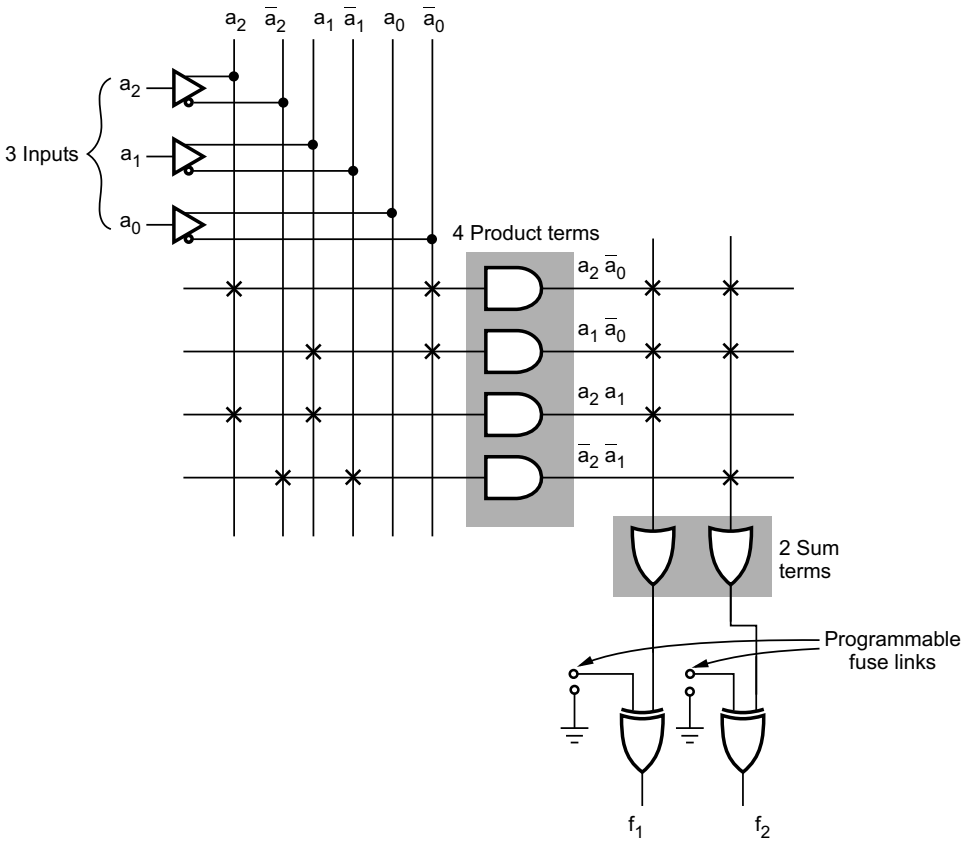


Fig. 10.11.29 Logic diagram

Example 10.11.13 *Design a BCD to Excess-3 code converter and implement using suitable PLA.*

Solution :

Step 1 : Derive the truth table of BCD to Excess-3 converter

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0

6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 10.11.6 Truth table for BCD to Excess-3 code converter

Step 2 : Simplify the Boolean functions for Excess-3 code

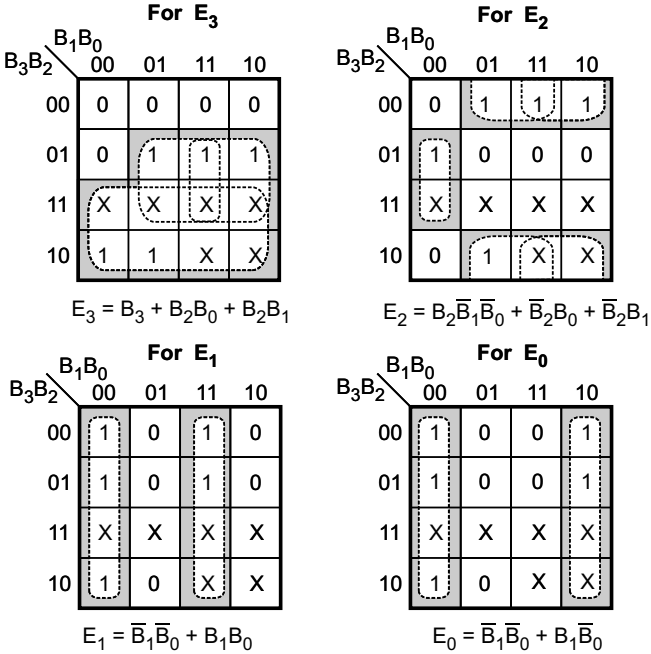

For E_3
For E_2
For E_1
For E_0
$E_3 = B_3 + B_2B_0 + B_2B_1$
$E_2 = B_2\overline{B}_1\overline{B}_0 + \overline{B}_2B_0 + \overline{B}_2B_1$
$E_1 = \overline{B}_1\overline{B}_0 + B_1B_0$
$E_0 = \overline{B}_1\overline{B}_0 + B_1\overline{B}_0$

Fig. 10.11.30

Step 3 : Write PLA program table

	Product terms	Inputs				Outputs			
		B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
B_3	1	1	–	–	–	1	–	–	–
$B_2\ B_0$	2	–	1	–	1	1	–	–	–
$B_2\ B_1$	3	–	1	1	–	1	–	–	–
$B_2\ \overline{B}_1\ \overline{B}_0$	4	–	1	0	0	–	1	–	–
$\overline{B}_2\ B_0$	5	–	0	–	1	–	1	–	–
$\overline{B}_2\ B_1$	6	–	0	1	–	–	1	–	–
$\overline{B}_1\ \overline{B}_0$	7	–	–	0	0	–	–	1	1
$B_1\ B_0$	8	–	–	1	1	–	–	1	–
$B_1\ \overline{B}_0$	9	–	–	1	0	–	–	–	1
						T	T	T	T	T/C

Table 10.11.7 PLA program table

Step 4 : Implementation

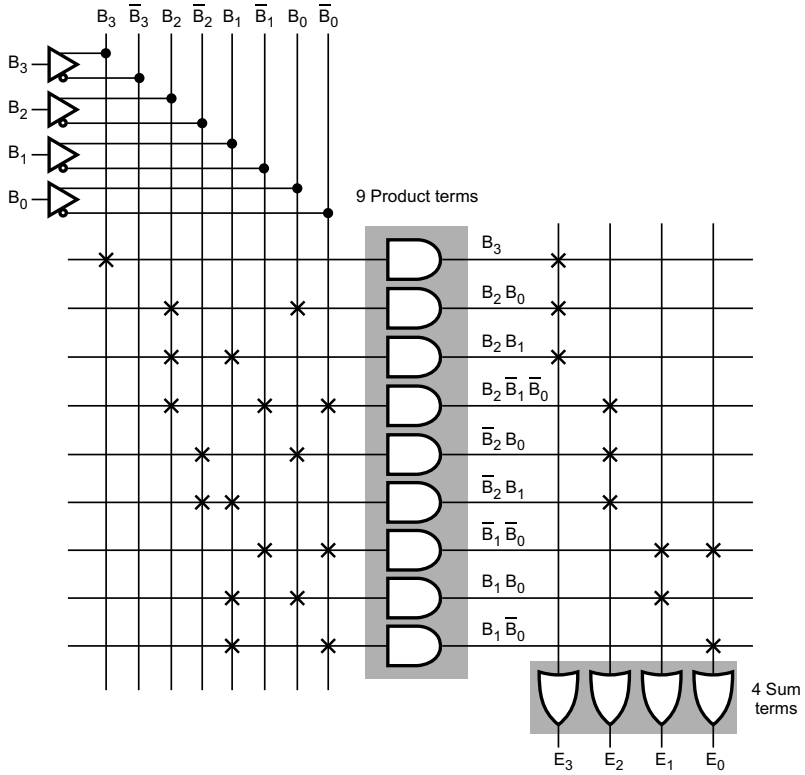


Fig. 10.11.31

Examples with Solutions

Example 10.11.14 *Design and implement 3-bit binary to gray code converter using PLA.*

Solution :

Step 1 : Derive the truth table for 3-bit binary to gray code converter

B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Table 10.11.8

Step 2 : Simplify the Boolean functions for gray code

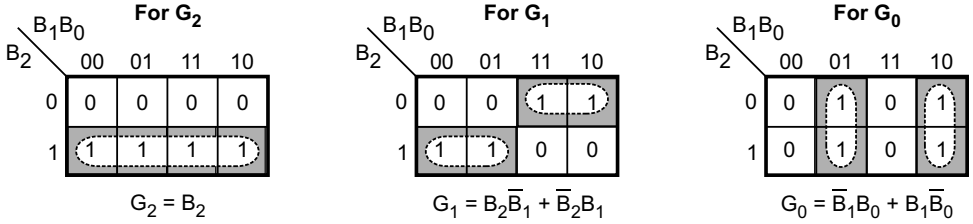


Fig. 10.11.32

Step 3 : Implementation

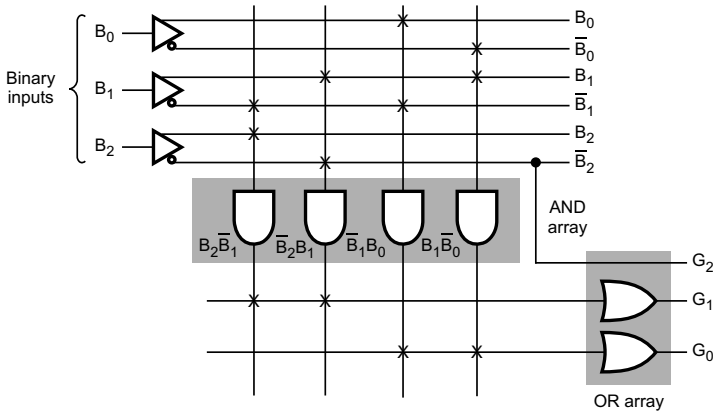


Fig. 10.11.33

Example 10.11.15 *Design a combinational circuit using PLA. The circuit accepts 3-bit number and generates an output binary number equal to square of input number.*

May-10, Marks 8

Solution :

Step 1 : Derive the truth table

Binary Input on address lines			Square of number on data lines					
A_2	A_1	A_0	F_5	F_4	F_3	F_2	F_1	F_0
0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	1	0	0	0	0	1	0	0
0	1	1	0	0	1	0	0	1
1	0	0	0	1	0	0	0	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	1	0	0
1	1	1	1	1	0	0	0	1

Table 10.11.9 PLA truth Table

Step 2 : Simplify Boolean functions of square output

For F_5

A_2 \ A_1A_0	00	01	11	10
0	0	0	0	0
1	0	0	1	1

$F_5 = A_2 A_1$

For F_4

A_2 \ A_1A_0	00	01	11	10
0	0	0	0	0
1	1	1	1	0

$F_4 = A_2 \overline{A}_1 + A_2 A_0$

For F_3

A_2 \ A_1A_0	00	01	11	10
0	0	0	1	0
1	0	1	0	0

$F_3 = \overline{A}_2 A_1 A_0 + A_2 \overline{A}_1 A_0$

For F_2

A_2 \ A_1A_0	00	01	11	10
0	0	0	0	1
1	0	0	0	1

$F_2 = A_1 \overline{A}_0$

For F_1

A_2 \ A_1A_0	00	01	11	10
0	0	0	0	0
1	0	0	0	0

$F_1 = 0$

For F_0

A_2 \ A_1A_0	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$F_0 = A_0$

Fig. 10.11.34

Step 3 : Implementation

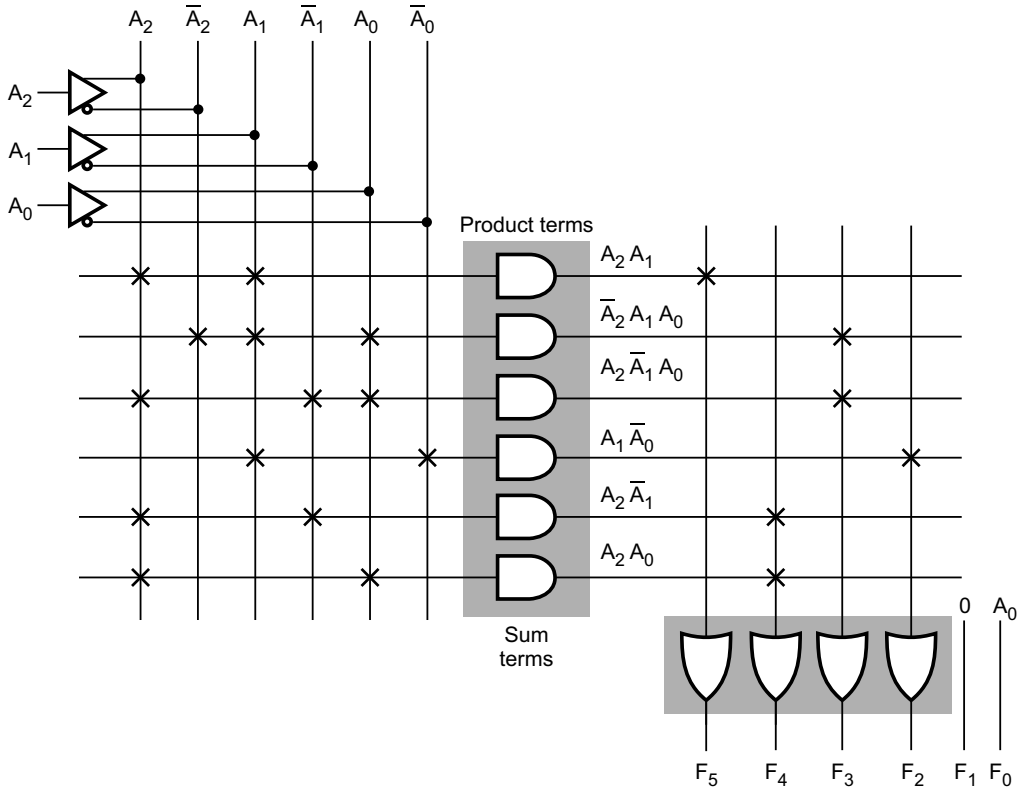


Fig. 10.11.35

Example 10.11.16 *Implement the following two Boolean functions with a PLA.*

$F_1 (A, B, C) = \sum(0, 1, 2, 4)$

$F_2 (A, B, C) = \sum(0, 5, 6, 7)$ **Dec.-07, Marks 16**

Solution :

Step 1 : Simplify the Boolean function

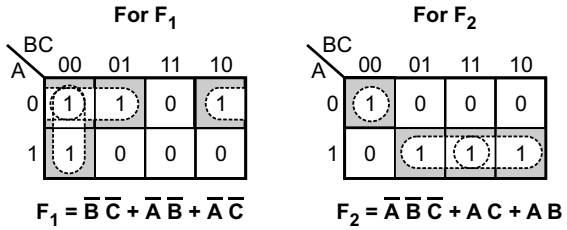


Fig. 10.11.36 K-map simplification

Step 2 : Implementation

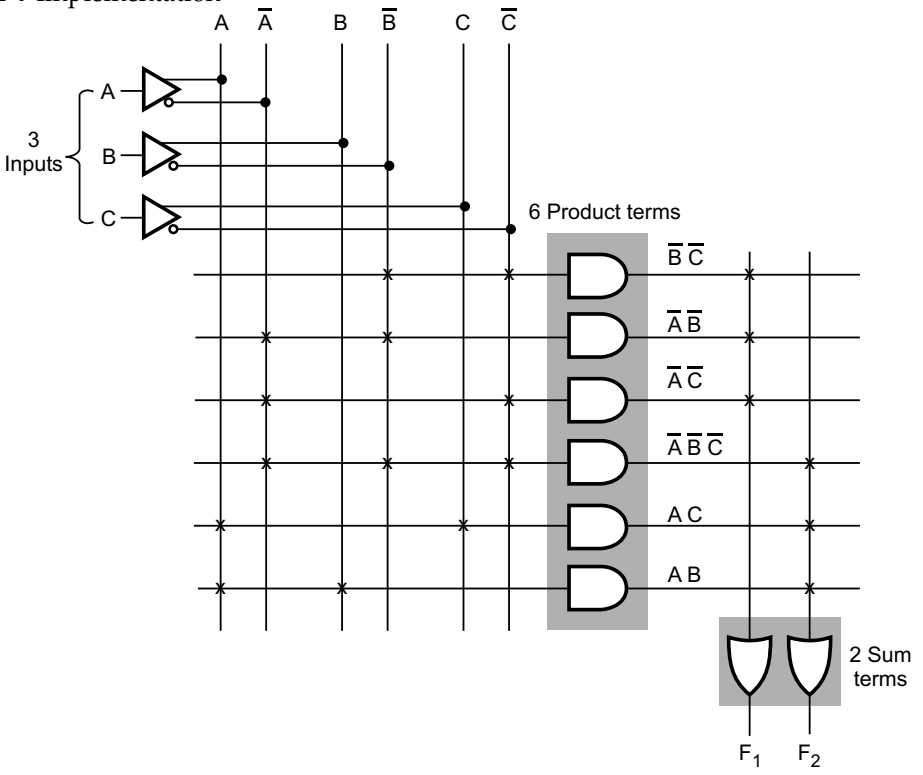


Fig. 10.11.37 Logic diagram

Example 10.11.17 *A combinational circuit is defined by functions.*

$F_1(A, B, C) = \sum(3, 5, 6, 7)$

$F_2 (A, B, C) = \sum(0, 2, 4, 7)$. *Implement the circuit with a PLA having three inputs, four product terms and two outputs.* **Dec.-08, 05, Marks 6**

Solution :

Step 1 : Simplify the Boolean functions

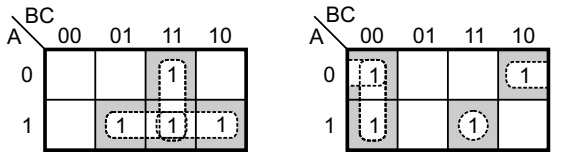


Fig. 10.11.38 K-map simplification

Here, we have 6 product terms so we check for complement functions.

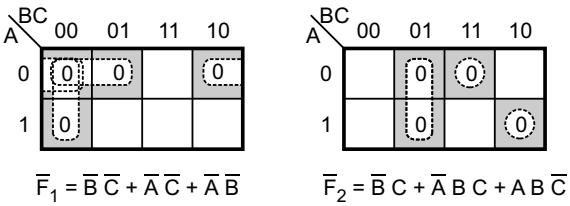


Fig. 10.11.39

If we take $\bar{f}_1$ and f_2 outputs of two functions, we need only four product terms since product terms $\overline{B}\,\overline{C}$ and $\overline{A}\,\overline{C}$ are common between them.

Step 2 : Implementation

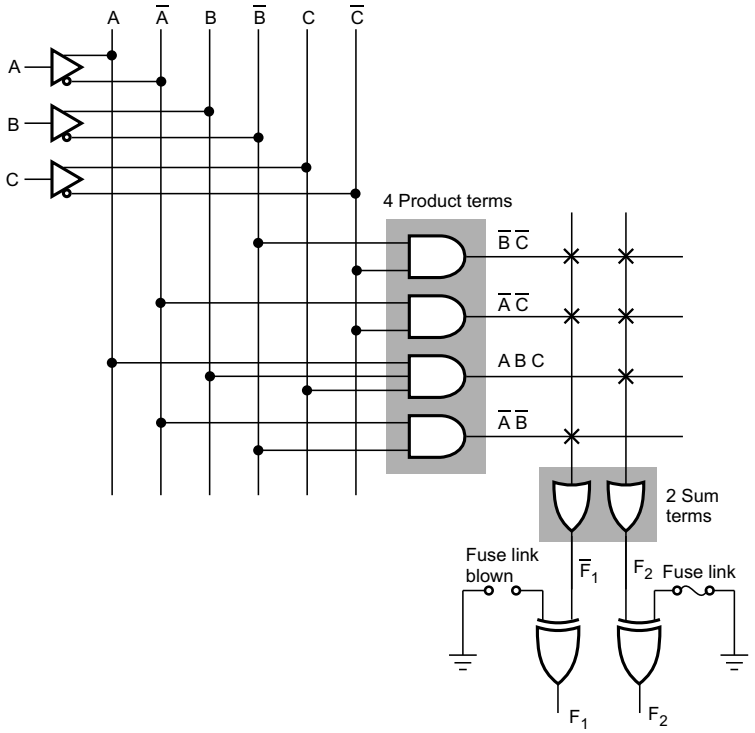


Fig. 10.11.40 Logic diagram

Example 10.11.18 *Implement the following functions using 3 input, 4 product term and 2 output PLA.*

$F_1 = AB' + AC + A'BC'$

$F_2 = (AC + BC)'$ **May-11, Marks 8**

Solution : Implementation

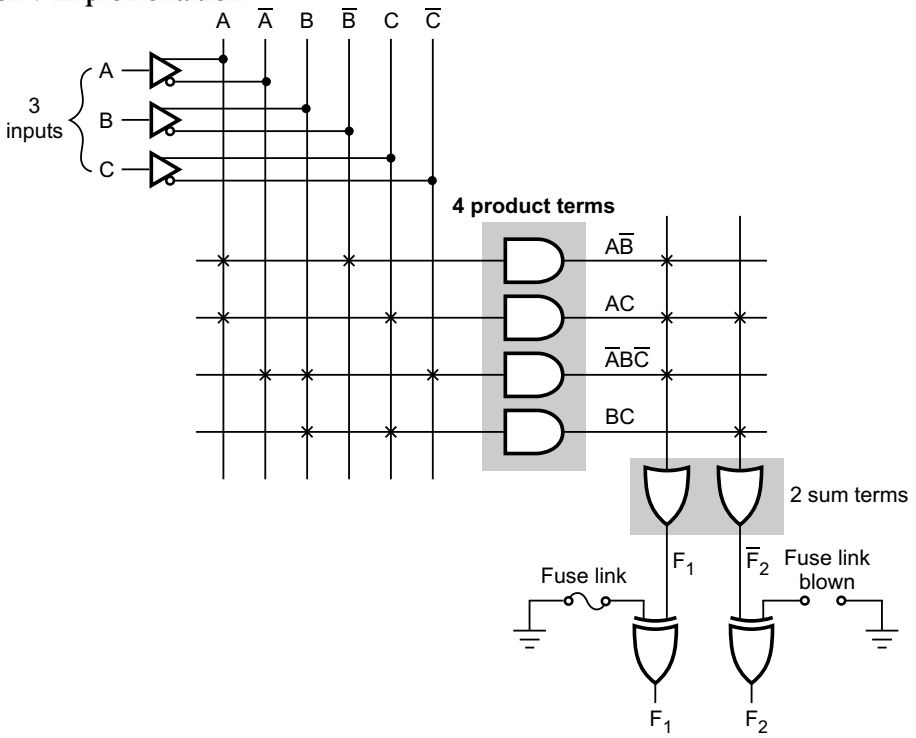


Fig. 10.11.41

Example 10.11.19 *Implement the following Boolean functions with a PLA.*

$F_1 (A, B, C) = \sum(0, 1, 2, 4)$

$F_2 (A, B, C) = \sum(0, 5, 6, 7)$

$F_3 (A, B, C) = \sum(0, 3, 5, 7)$ **Dec.-09, Marks 16**

Solution : K-map simplification

For F_1

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$F_1 = \overline{B}\,\overline{C} + \overline{A}\,\overline{B} + \overline{A}\,\overline{C}$

For F_2

A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$F_2 = \overline{A}\,\overline{B}\,\overline{C} + A\,C + A\,B$

For F_3

A \ BC	00	01	11	10
0	1	0	1	0
1	0	1	1	0

$F_3 = \overline{A}\,\overline{B}\,\overline{C} + B\,C + A\,C$

Fig. 10.11.42

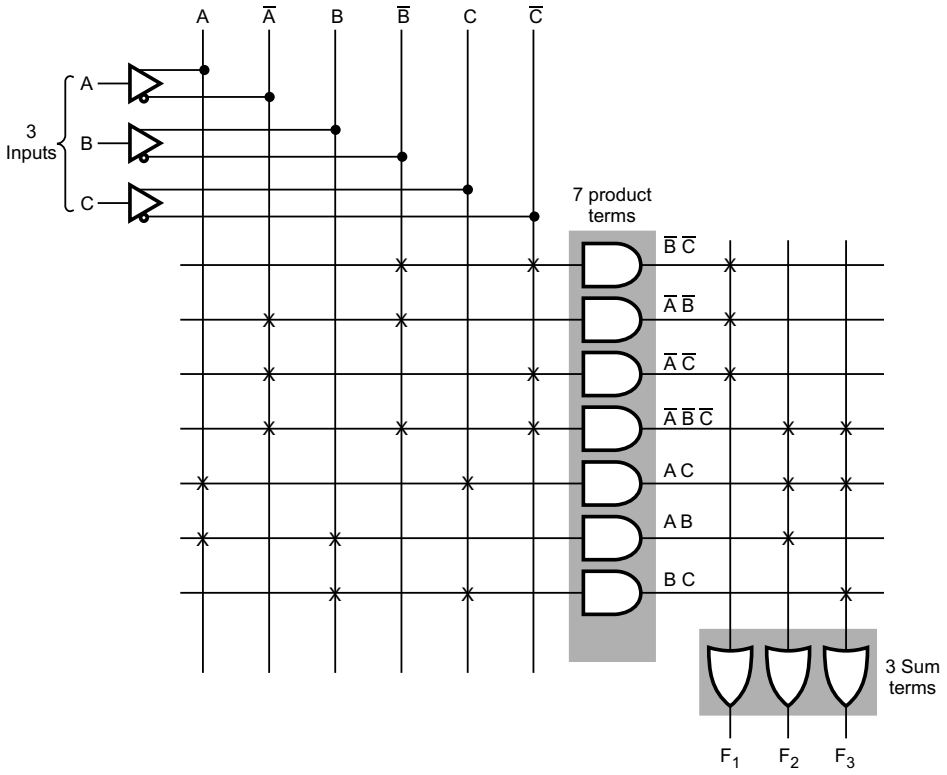


Fig. 10.11.43

Example 10.11.20 *A combinational circuit is defined as the functions*

F1 = AB'C'+AB'C+ABC

F2 = A'BC+AB'C+ABC Implement the digital circuit with a PLA having 3 inputs, 3 product terms, and 2 outputs. **May-10, Marks 8**

Solution : Let us simplify the functions :

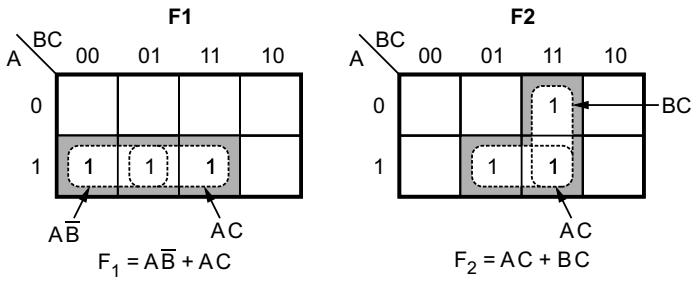


Fig. 10.11.44

For further solution refer example 10.11.10.

Example 10.11.21 *Tabulate the PAL programmable table for the four Boolean functions listed below.*

$A(x, y, z) = \sum m(1, 2, 4, 6)$

$B(x, y, z) = \sum m(0, 1, 6, 7)$

$C(x, y, z) = \sum m(2, 6)$

$D(x, y, z) = \sum m(1, 2, 3, 5, 7)$ **Dec.-11, Marks 6**

Solution : Let us simplify for minimum number of minterms.

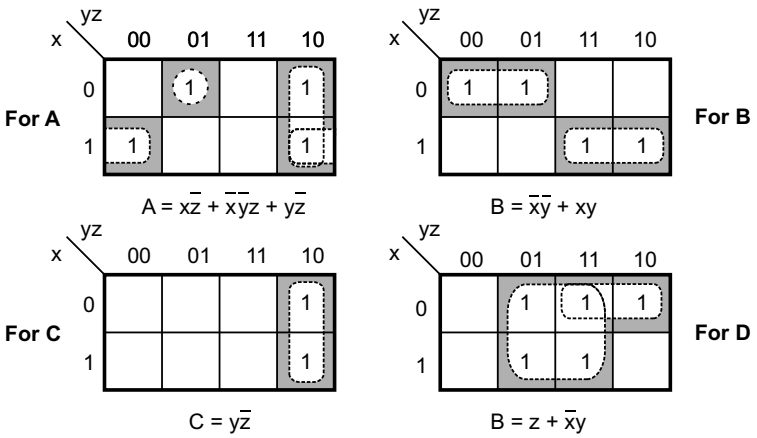


Fig. 10.11.45

Product terms	AND Inputs				Outputs
	x	y	z	C	
1	–	–	–	1	$A = C + x\bar{z} + \bar{x}\bar{y}z$
2	1	–	0	–	
3	0	0	1	–	
4	0	0	–	–	$B = \bar{x}\bar{y} + xy$
5	1	1	–	–	
6	–	–	–	–	
7	–	1	0	–	$C = y\bar{z}$
8	–	–	–	–	
9	–	–	–	–	
10	–	–	1	–	$D = z + \bar{x}y$
11	0	1	–	–	
12	–	–	–	–	

Table 10.11.10 PAL program table

Example 10.11.22 *Draw a PLA circuit to implement the functions*
$F_1 = A'B + AC' + A'BC'$; $F_2 = (AC + AB + BC)'$ **Dec.-10, Marks 8**

Solution :

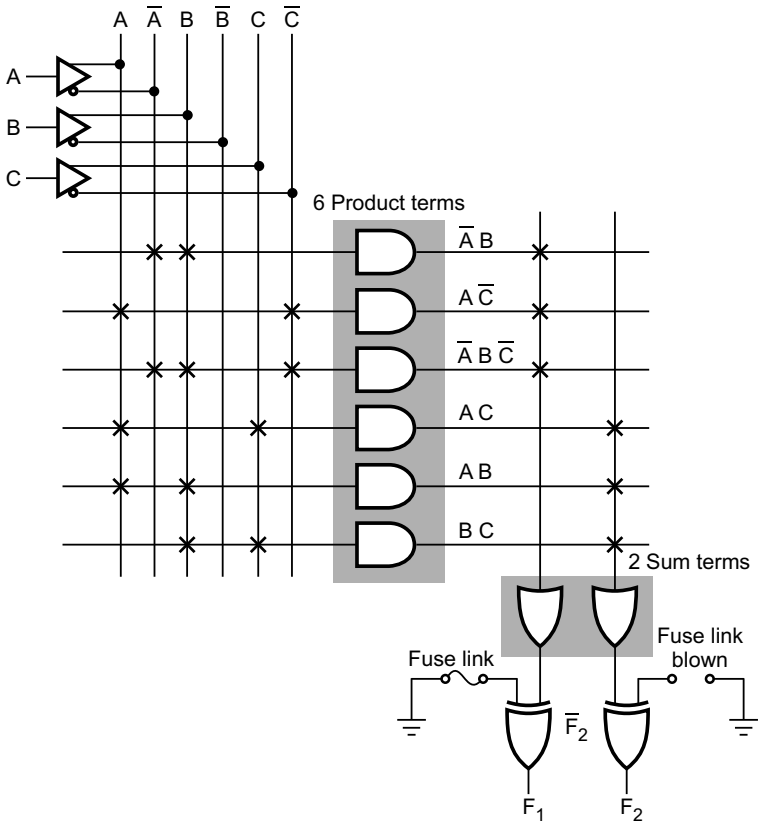


Fig. 10.11.46

10.11.3 PAL (Programmable Array Logic)

We have seen that PLA is a device with a programmable AND array and programmable OR array. However, PAL programmable array logic is a programmable logic device with a fixed OR array and a programmable AND array. Because only AND gates are programmable, the PAL is easier to program, but is not as flexible as the PLA. Fig. 10.11.48 shows the array logic of a typical PAL. It has four inputs and four outputs. Each input has buffer and an inverter gate. It is important to note that two gates are shown with one composite graphic symbol with normal and complement outputs. There are four sections. Each section has three programmable AND gates and one fixed OR gate. The output of section 1 is connected to a buffer-inverter gate and then fed back into the inputs of the AND gates, through fuses. This allows the logic designer to feed

an output function back as an input variable to create a new function. Such PALs are referred to as **Programmable I/O PALs**. Refer Fig. 10.11.47 on next page.

The commercial PAL devices has more gates than the one shown in Fig. 10.11.47 A typical PAL integrated circuit may have eight inputs, eight outputs, and eight sections, each consisting of an eight wide AND-OR array.

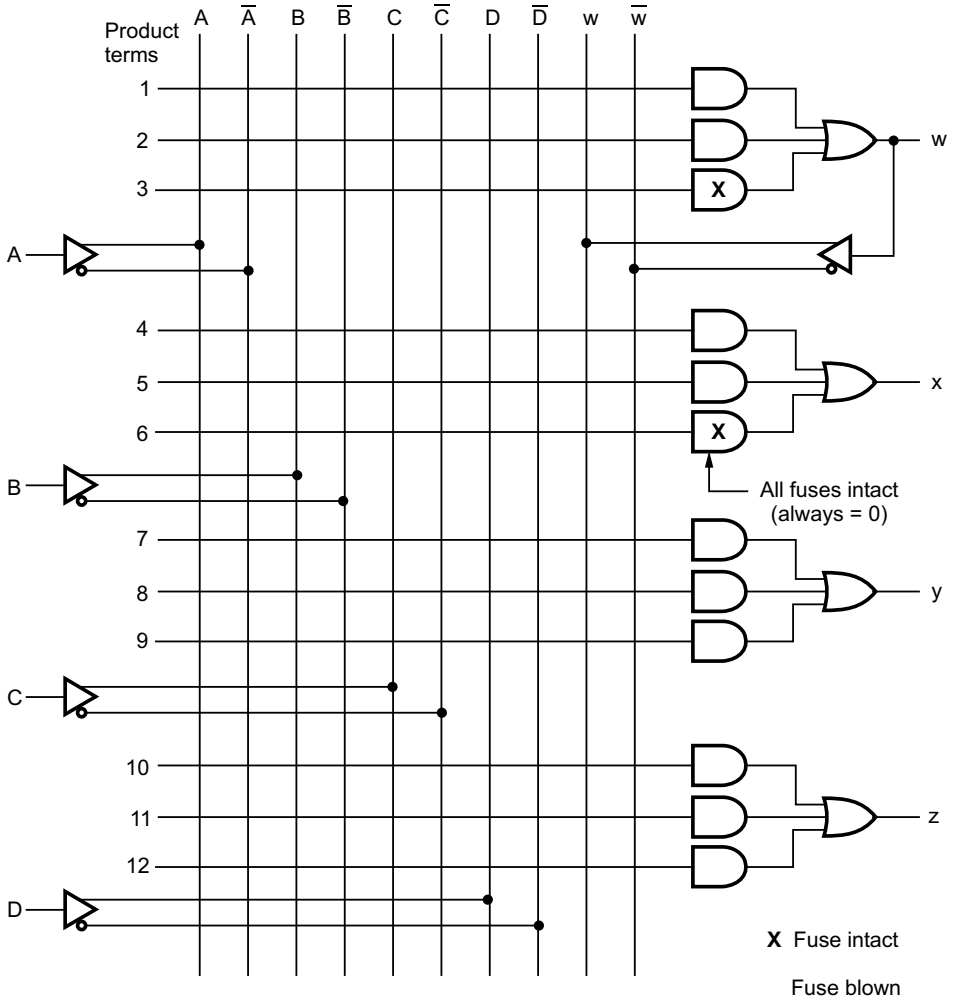


Fig. 10.11.47 Array logic for typical PAL

10.11.3.1 Implementation of Combinational Logic Circuit using PAL

Let us see the implementation of a combinational circuit using PAL with the help of examples.

Examples for Understanding

Example 10.11.23 *Implement the following Boolean functions using PAL.*

$w\ (A, B, C, D) = \sum m\ (0, 2, 6, 7, 8, 9, 12, 13)$

$x\ (A, B, C, D) = \sum m\ (0, 2, 6, 7, 8, 9, 12, 13, 14)$

$y\ (A, B, C, D) = \sum m\ (2, 3, 8, 9, 10, 12, 13)$

$z\ (A, B, C, D) = \sum m\ (1, 3, 4, 6, 9, 12, 14)$

Solution :

Step 1 : Simplify the four functions

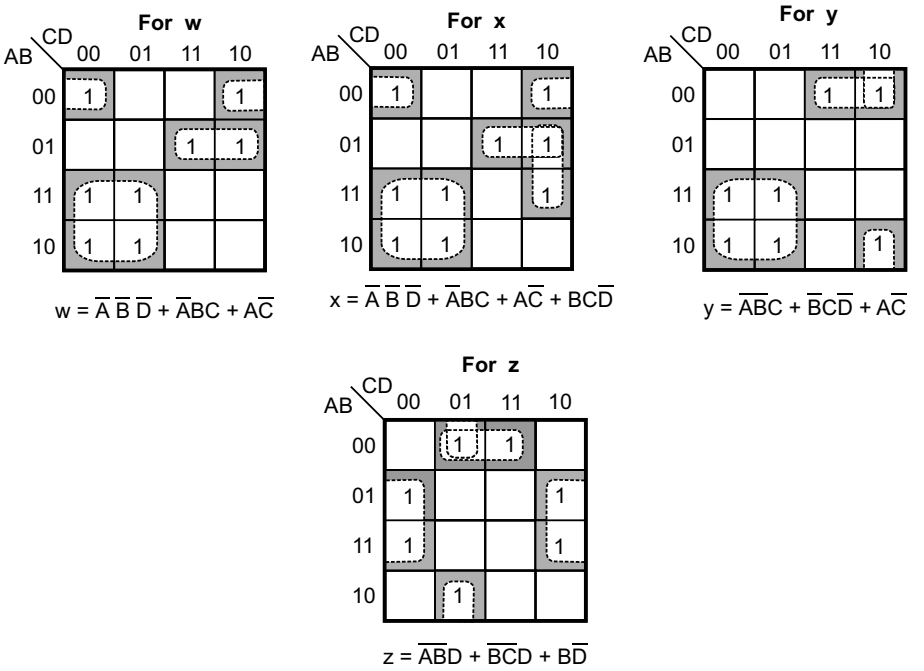


Fig. 10.11.48 K-map simplification

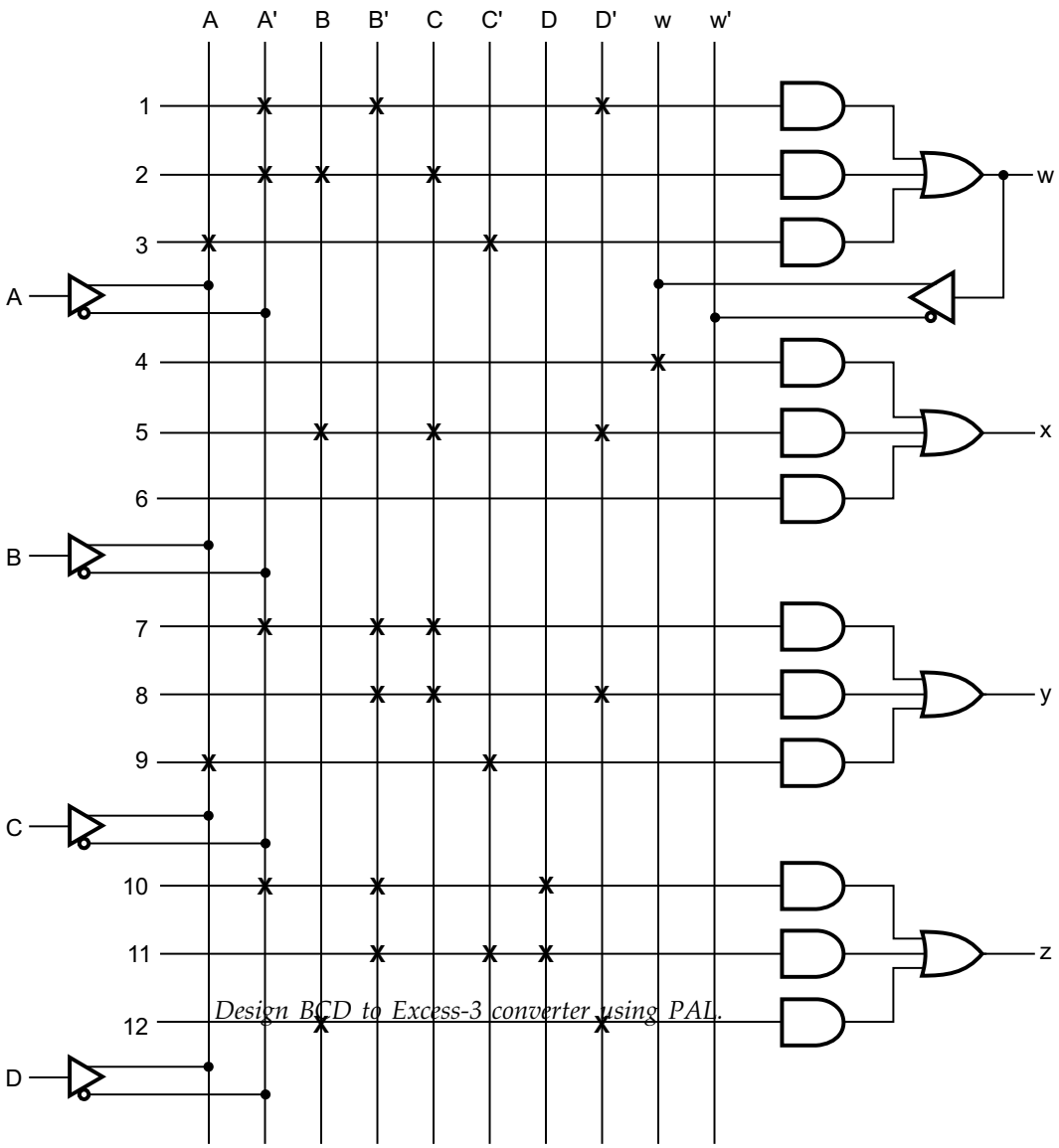
Note that function x has four product terms. Three of them are equal to w. Therefore we can write $x = w + BC\overline{D}$.

Step 2 : Implementation

In the last section we have seen the PLA program table. The program table for PAL is similar to PLA program table. Table 10.11.11 shows PAL program table with product terms, AND inputs and outputs.

Product term	AND Inputs					Outputs
	A	B	C	D	w	
1	0	0	–	0	–	$w = \overline{A}\,\overline{B}\,\overline{D} + \overline{A}\,BC + A\,\overline{C}$
2	0	1	1	–	–	
3	1	–	0	–	–	
4	–	–	–	–	1	$x = w + BC\overline{D}$
5	–	1	1	0	–	
6	–	–	–	–	–	
7	0	0	1	–	–	$y = \overline{A}\,\overline{B}\,C + \overline{B}\,C\,\overline{D} + A\,\overline{C}$
8	–	0	1	0	–	
9	1	–	0	–	–	
10	0	0	–	1	–	$z = \overline{A}\,\overline{B}\,D + \overline{B}\overline{C}\,D + B\,\overline{D}$
11	–	0	0	1	–	
12	–	1	–	0	–	

Table 10.11.11 PAL program table



Design BCD to Excess-3 converter using PAL.

Fig. 10.11.49 Logic diagram

Example 10.11.24

Solution :

Step 1 : Derive the truth table of BCD to Excess-3 converter

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

Table 10.11.12 Truth table for BCD to Excess-3 code converter

Step 2 : Simplify the Boolean functions for Excess-3 code outputs.

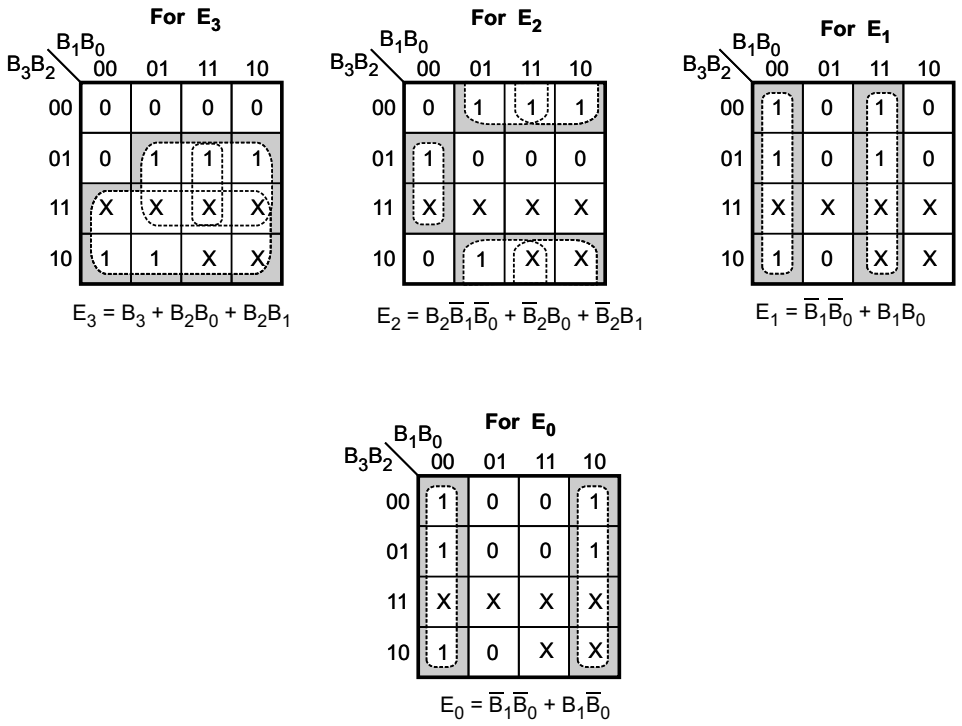


Fig. 10.11.50 K-map simplification

Step 3 : Implementation

	Product terms	Inputs				Outputs				
		B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0	
B_3	1	1	–	–	–	1	–	–	–	
$B_2\ B_0$	2	–	1	–	1	1	–	–	–	
$B_2\ B_1$	3	–	1	1	–	1	–	–	–	
$B_2\ \overline{B}_1\ \overline{B}_0$	4	–	1	0	0	–	1	–	–	
$\overline{B}_2\ B_0$	5	–	0	–	1	–	1	–	–	
$\overline{B}_2\ B_1$	6	–	0	1	–	–	1	–	–	
$\overline{B}_1\ \overline{B}_0$	7	–	–	0	0	–	–	1	1	
$B_1\ B_0$	8	–	–	1	1	–	–	1	–	
$B_1\ \overline{B}_0$	9	–	–	1	0	–	–	–	1	
						T	T	T	T	T/C

Table 10.11.13 PAL program table

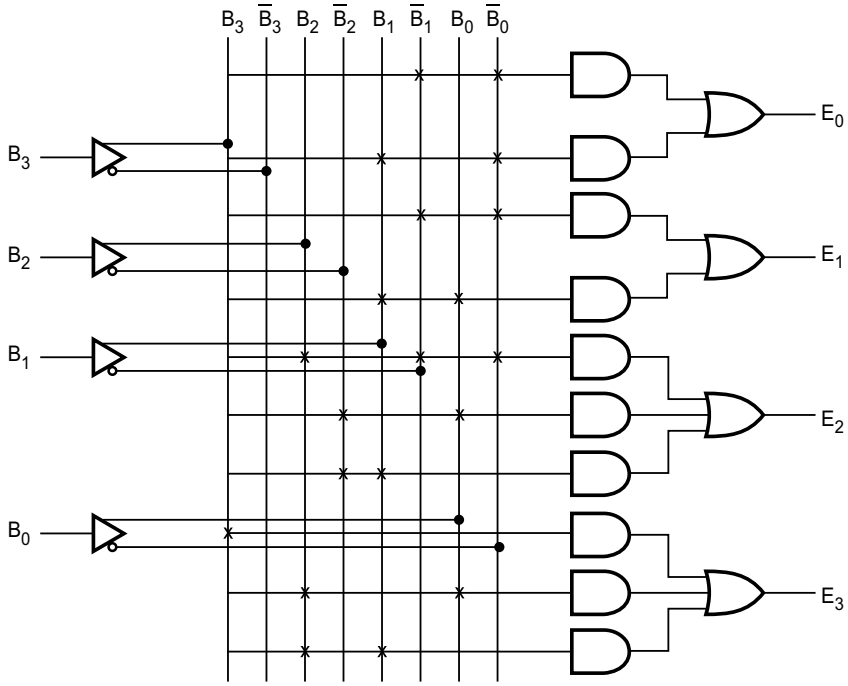


Fig. 10.11.51 Logic diagram

Example 10.11.25 *Generate the following Boolean functions with a PAL with 4 inputs and 4 outputs.*

$Y_3 = \overline{A}B\overline{C}D + \overline{A}BC\overline{D} + AB\overline{C}D, \quad Y_2 = \overline{A}BC\overline{D} + \overline{A}BCD + ABCD$

$Y_1 = \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}$, $Y_0 = ABCD$ **Dec.-04, Marks 8**

Solution :

Step 1 : Simplify the Boolean functions

$$Y_3 = \overline{A}\ B\ \overline{C}\ D + \overline{A}\ B\ C\ \overline{D} + A\ B\ \overline{C}\ D = (\overline{A} + A)\ B\ \overline{C}\ D + \overline{A}\ B\ C\ \overline{D}$$

$$= B\ \overline{C}\ D + \overline{A}\ B\ C\ \overline{D}$$

$$Y_2 = \overline{A}\ B\ C\ \overline{D} + \overline{A}\ B\ C\ D + A\ B\ C\ D = \overline{A}\ B\ C\ (\overline{D} + D) + (\overline{A} + A)\ (B\ C\ D)$$

$$= \overline{A}\ B\ C + B\ C\ D$$

$$Y_1 = \overline{A}\ B\ \overline{C} + \overline{A}\ B\ C + A\ \overline{B}\ C + A\ B\ \overline{C}$$

$$= \overline{A}\ B\ (\overline{C} + C) + A\ \overline{B}\ C + (\overline{A} + A)\ B\ \overline{C} = \overline{A}\ B + A\ \overline{B}\ C + B\ \overline{C}$$

$$Y_0 = A\ B\ C\ D$$

Step 2 : Implementation

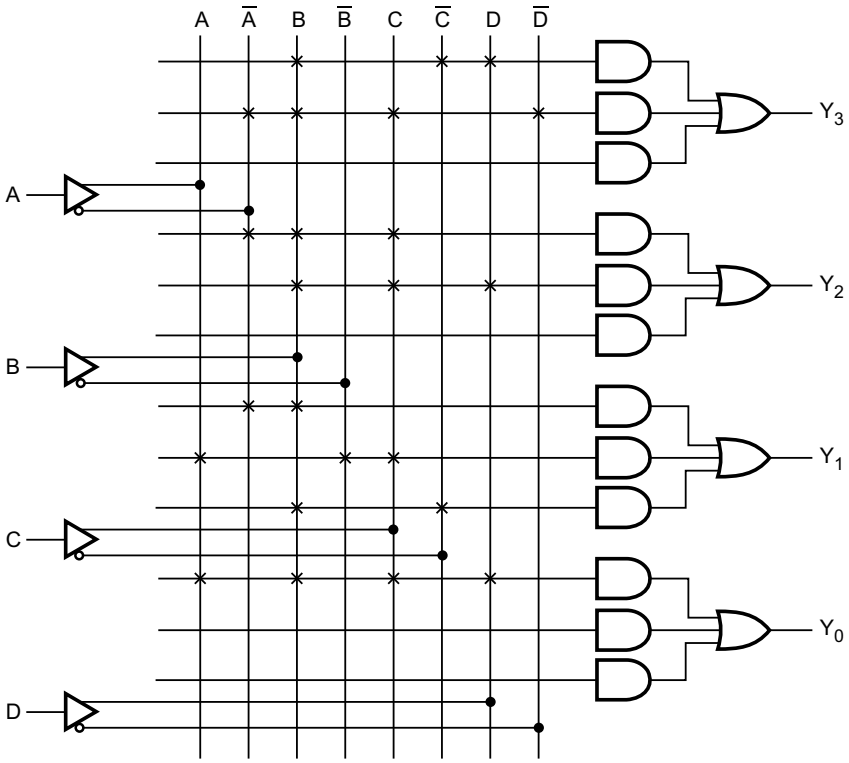


Fig. 10.11.52 Logic diagram

Example for Practice

Example 10.11.26 *A combinational logic circuit is defined by the following function.* $f_1(a,b,c) = \Sigma\,(0,1,6,7)$, $f_2(a,b,c) = \Sigma\,(2,3,5,7)$. *Implement the circuit with a PAL having three inputs, three product terms and two outputs.*

May-05, Marks 10

10.11.4 Comparison between PROM, PLA and PAL

Sr. No.	PROM	PLA	PAL
1.	AND array is fixed and OR array is programmable.	Both AND and OR arrays are programmable.	OR array is fixed and AND array is programmable.
2.	Cheaper and simple to use.	Costliest and complex than PAL and PROMs.	Cheaper and simpler.
3.	All minterms are decoded.	AND array can be programmed to get desired minterms.	AND array can be programmed to get desired minterms.
4.	Only Boolean functions in standard SOP form can be implemented using PROM.	Any Boolean functions in SOP form can be implemented using PLA.	Any Boolean functions in SOP form can be implemented using PLA.

10.11.5 Registered PALs

The PAL device discussed in the previous section does not contain any memory elements. Such PALs are useful to implement combinational logic circuits. Hence, PAL devices which do not contain any memory elements such as flip-flops are called **combinational PALs.**

Some PALs have flip-flops at the output. The flip-flops store the output. The several flip-flops at the output form a register and because these outputs have tri-state buffers, the output of register can be controlled. Such PALs are referred to as **registered PALs.** Refer Fig. 10.11.53. As shown in the Fig. 10.11.53, the flip-flop outputs are made available as an input to be used in the generation of additional product terms. This feedback feature allows PALs to be used to design counters, registers and sequential controllers.

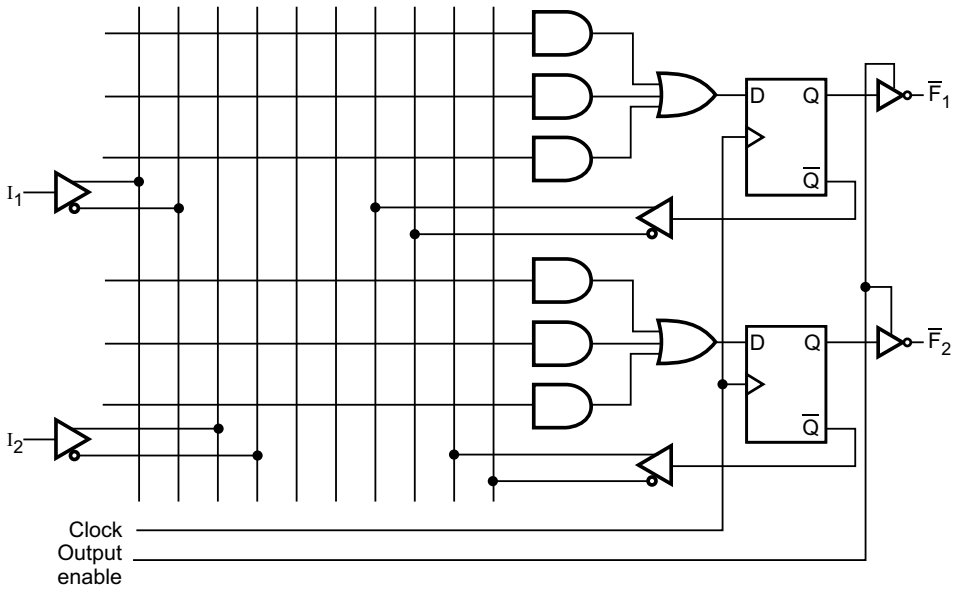


Fig. 10.11.53 Structure of registered PAL

Example 10.11.27 *Implement the following state diagram using registered output PAL.*

Solution : The state table for above state diagram is as given below.

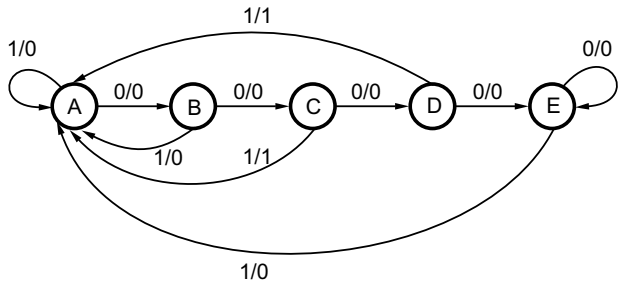


Fig. 10.11.54

Present state	Next state		Output	
	X = 0	X = 1	X = 0	X = 1
A	B	A	0	0
B	C	A	0	0
C	D	A	0	1
D	E	A	0	1
E	E	A	0	0

Table 10.11.14

Let us assign states as follows

A = 000,

B = 001,

C = 010,

D = 011,

E = 100

We need three D flip-flops with output Q_A (MSB), Q_B and Q_C (LSB).

	Present state			Next state			Output
X	**Q_A**	**Q_B**	**Q_C**	**Q_{A+1}**	**Q_{B+1}**	**Q_{C+1}**	**Y**
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0
1	0	1	0	0	0	0	1
1	0	1	1	0	0	0	1
1	1	0	0	0	0	0	0

From the above table we have excitation equation for flip-flop as follows.

$$D_A = \overline{X}\, Q_A\, \overline{Q}_B\, \overline{Q}_C + \overline{X}\, \overline{Q}_A\, Q_B\, Q_C$$

$$D_B = \overline{X}\, \overline{Q}_A\, \overline{Q}_B\, Q_C + \overline{X}\, \overline{Q}_A\, Q_B\, \overline{Q}_C$$

$$D_C = \overline{X}\, \overline{Q}_A\, \overline{Q}_B\, \overline{Q}_C + \overline{X}\, \overline{Q}_A\, Q_B\, \overline{Q}_C$$

$$Y = X\, \overline{Q}_A\, Q_B\, \overline{Q}_C + X\, \overline{Q}_A\, Q_B\, Q_C$$

From the above excitation equations for D flip-flop we can say that the registered PAL should have : 4 inputs, 4 outputs, 4 sections with two AND gates and one OR gate in each section.

Logic diagram

The PAL 16R4 satisfies the requirements of PAL. The Fig. 10.11.55 shows the circuit implementation using PAL 16R4. As shown in the Fig. 10.11.55, each section of PAL16R4 has 8 product terms, available for each D excitation input. Therefore, the simplification of the excitation equations from state table would be necessary only if the number of product terms for any D input exceeds eight product terms. In our example, we need only two product terms for each D excitation input. Thus simplification of excitation equations is not required.

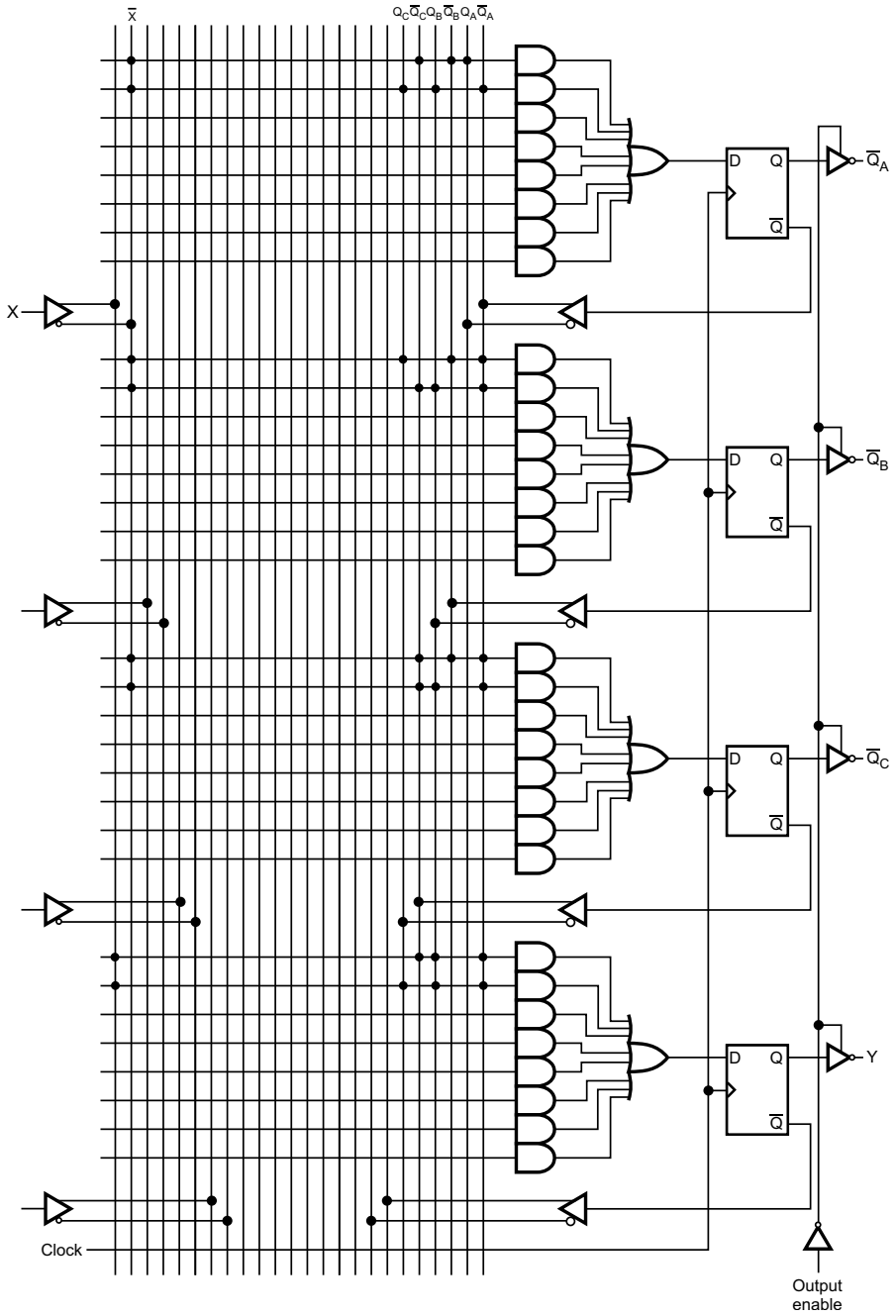

$\overline{X}$
$Q_C\overline{Q}_C Q_B\overline{Q}_B Q_A\overline{Q}_A$
X
D
Q
$\overline{Q}$
$\overline{Q}_A$
$\overline{Q}_B$
$\overline{Q}_C$
Y
Clock
Output enable

Fig. 10.11.55

Example 10.11.28 *Implement 3-bit updown counter using registered PAL.* **Dec.-12, Marks 8**

Solution : Table 10.11.15 shows the state table for 3-bit up/down synchronous counter.

Input $\overline{UP/DOWN}$ (M)	Present state			Next state		
X	Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	1
0	1	0	1	1	0	0
0	1	1	0	1	0	1
0	1	1	1	1	1	0
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	0	0	0

Table 10.11.15 3-bit up/down synchronous counter

From the state table we can see that the excitation equation for each D flip-flop contains eight minterms. This is because for eight minterms output of D flip-flop is logic 1. These eight minterms for each D flip-flop can be implemented by PAL16R4 as shown in the Fig. 10.11.56. (See Fig. 10.11.56 on next page.)

10.11.6 Configurable PALs

In recent years, configurable (sometimes called *generic*) device architectures have become extremely popular. These devices simplify procurement, qualification and inventory requirements by replacing a large number of simpler PAL type devices with a "one size fits all" device. In addition, their flexible architectures allow designs to be implemented that are challenging or simply impossible for the simpler PAL devices to handle.

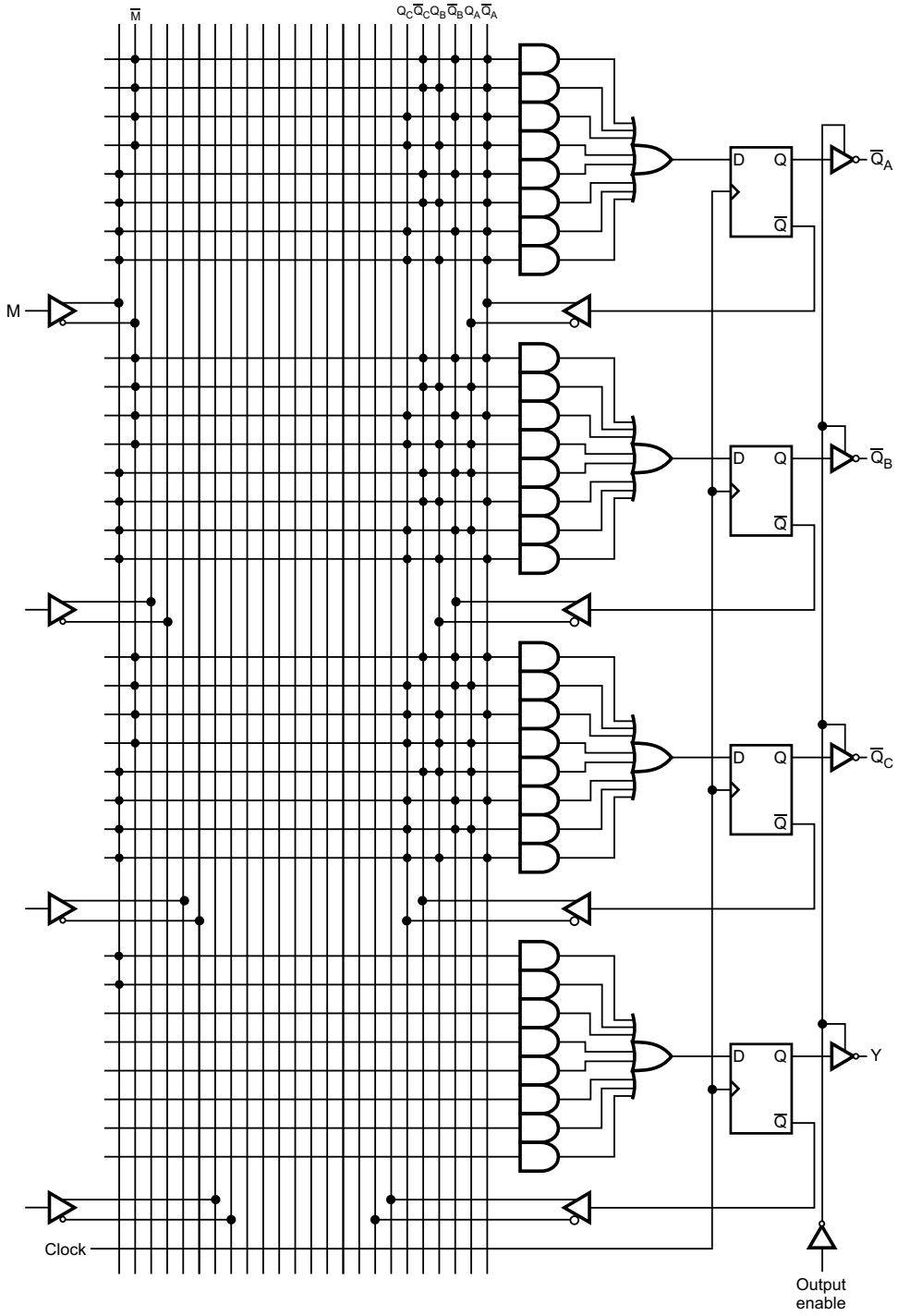


Fig. 10.11.56 Implementation of 3-bit up/down counter using PAL16R4

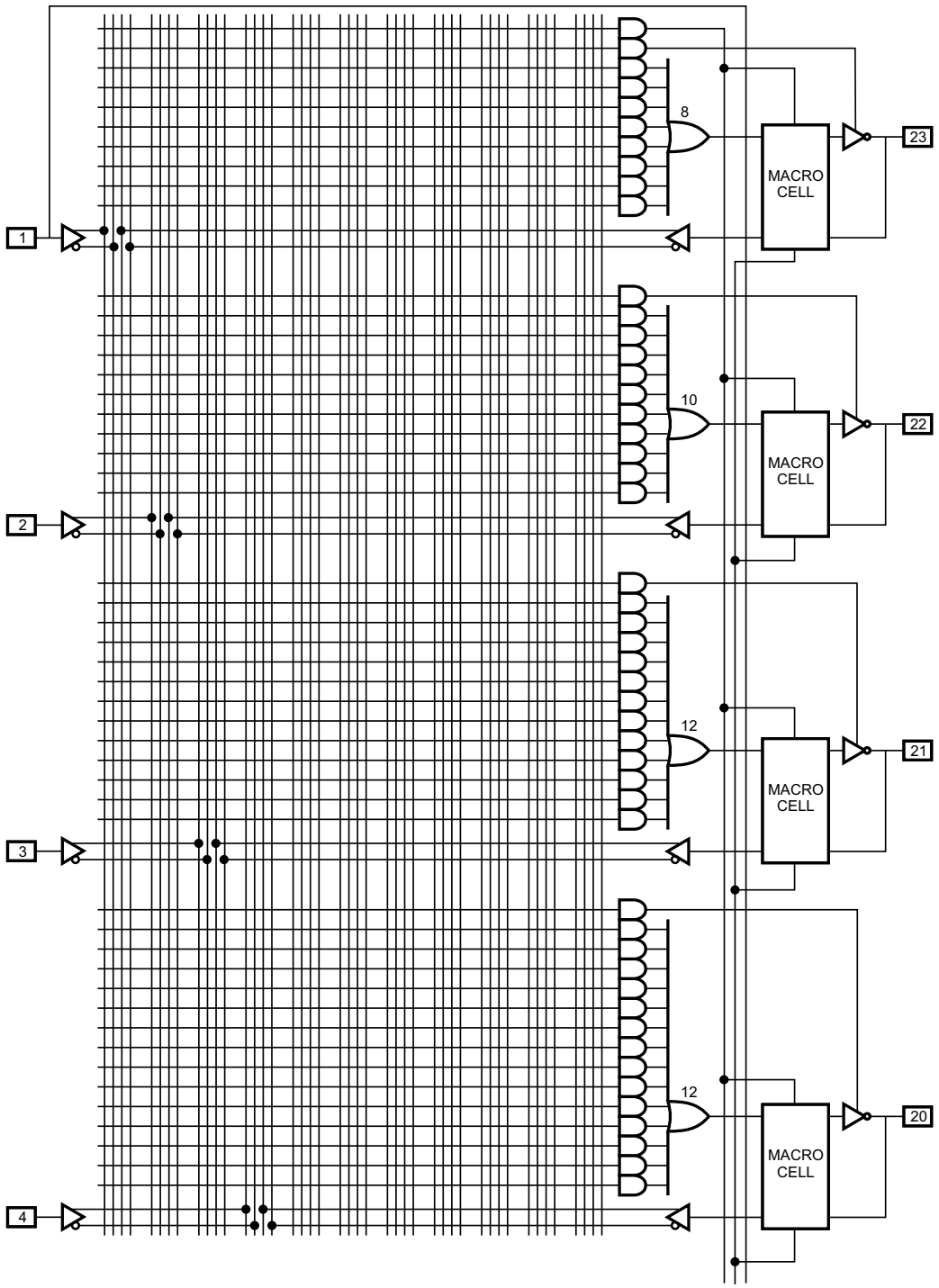


Fig. 10.11.57 **Continued …**

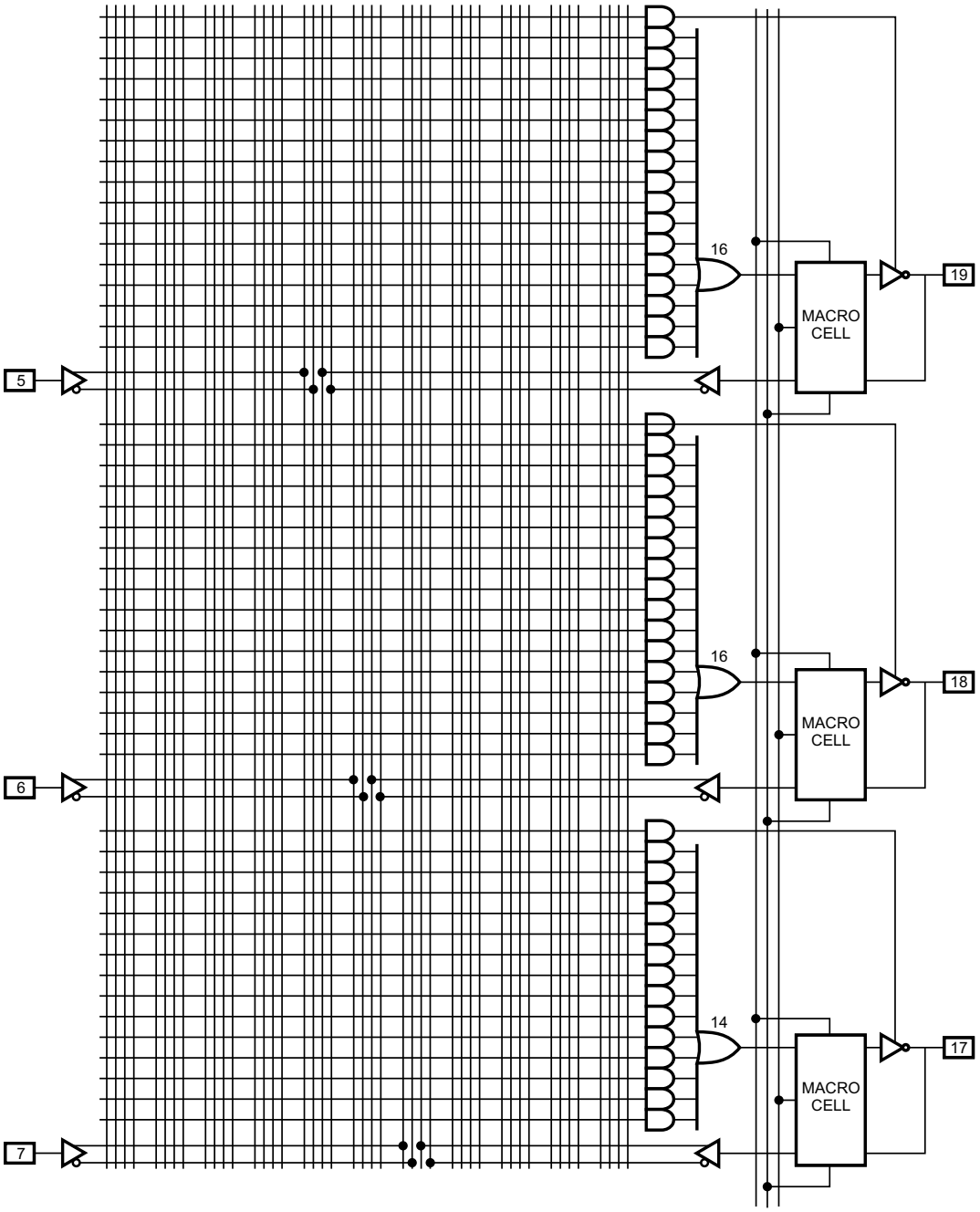


Fig. 10.11.57 **Continued …**

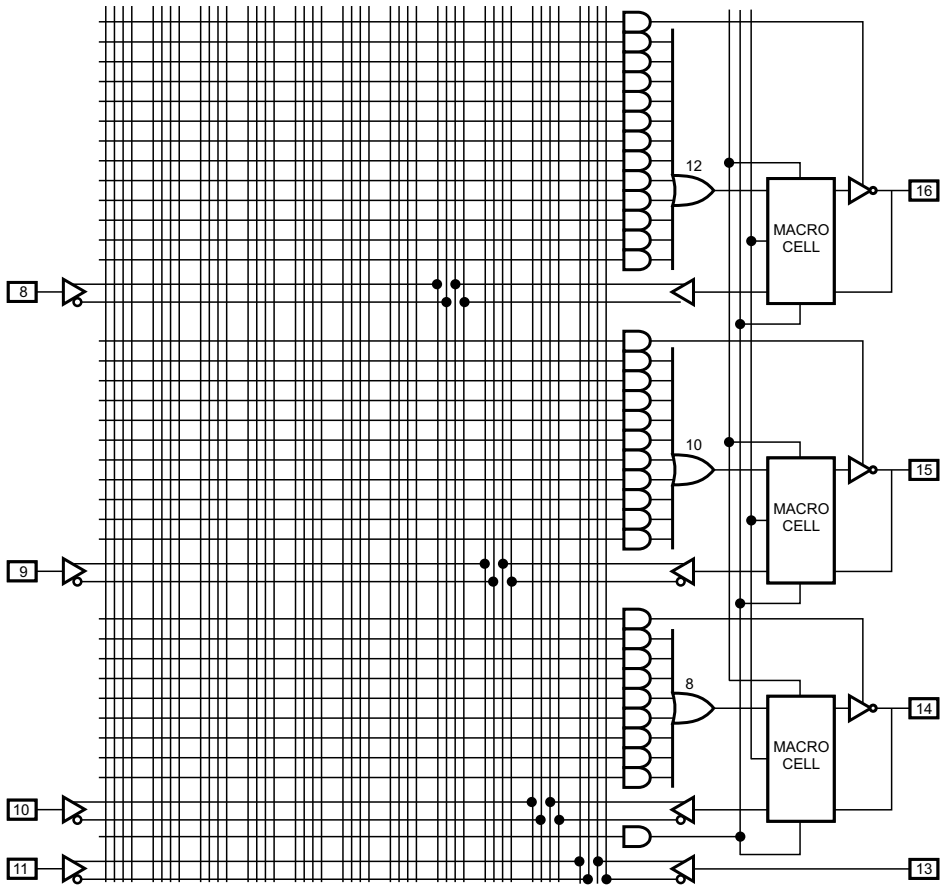


Fig. 10.11.57

This architectural flexibility is provided by equipping the device with a variety of configuration fuses separate from those found in the programmable AND array. The 22V10 is one such device and was designed by AMD to be a replacement for all of the 24-pin PALs. To meet this requirement, the 22V10 was designed with configurable outputs. These outputs are enhanced with special circuitry called **output macrocells.** Output macrocells are found on ten of the 22V10's pins, as shown in Fig. 10.11.53.

The 22V10 has a total of twelve dedicated inputs, one of which (pin 1) also functions as the common clock input to the edge-triggered D type flip-flop of each output macrocell. Any of the 22V10's ten output pins can be used as inputs, so the device is capable of supporting applications requiring upto 22 inputs (of course, if you use all ten I/O pins as dedicated inputs, there is no way to observe the results of your efforts, since there will be no output pins left).

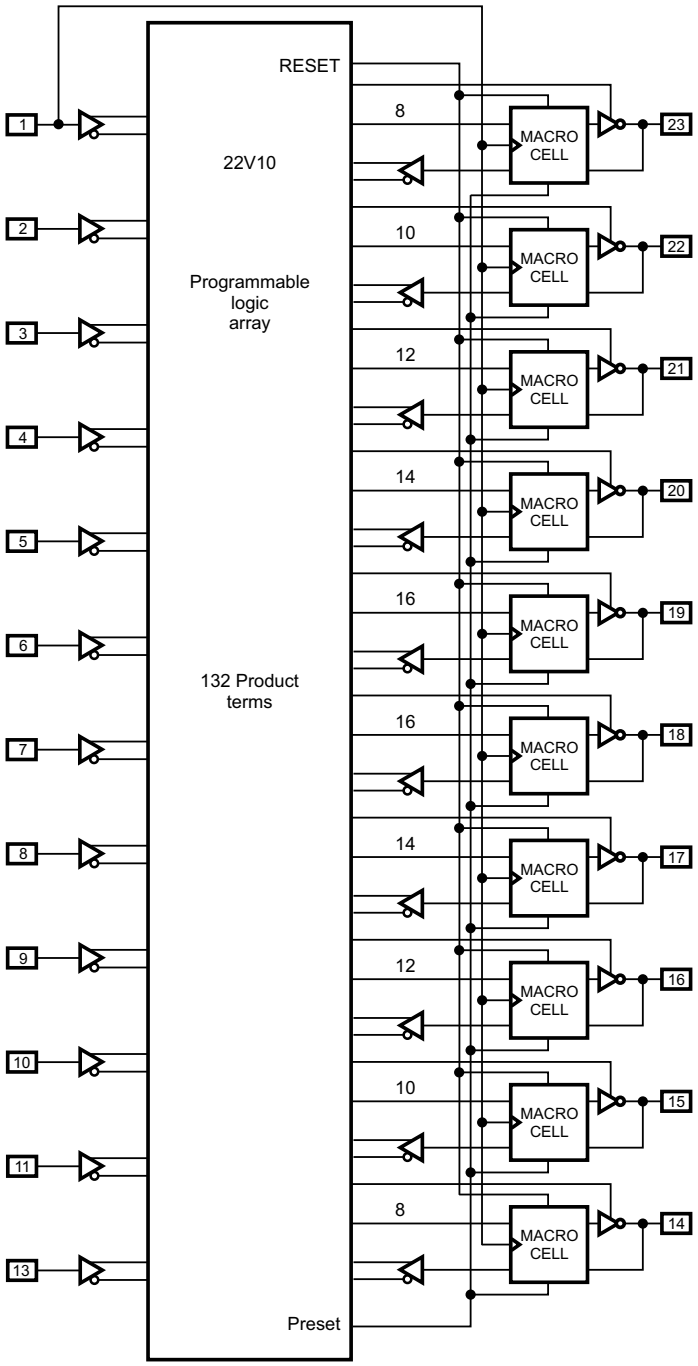


Fig. 10.11.58 Block diagram of 22V10

We will examine the structure of the individual output macrocells momentarily, but first, notice that the number of product terms available to the various OR gates in the device differs. The OR gate associated with pins 18 and 19, in fact, have sixteen product terms each available. This means that logic functions of significantly more complexity can be implemented in the 22V10. Fig. 10.11.57 illustrates the 22V10 in block diagram form. (See Fig. 10.11.58 on previous page.)

As mentioned earlier the 22V10 has ten output macrocells, all of which are identical. Fig. 10.11.59 shows the construction of one of these output macrocells. Each macrocell contains a positive edge-triggered D type flip-flop and a three configurable multiplexers. The two fuses S_0 and S_1 that control the multiplexers can be configured in four different ways, as shown in Fig. 10.11.60. The Fig. 10.11.60 does not show the fuse interconnections multiplexers and pull-up resistors. But it shows the effective circuit at various combinations of S_0 and S_1 fuses. From the Fig. 10.11.60 we can observed that this configurable PAL can be used as registered PAL or combinational PAL with inverted or non-inverted outputs.

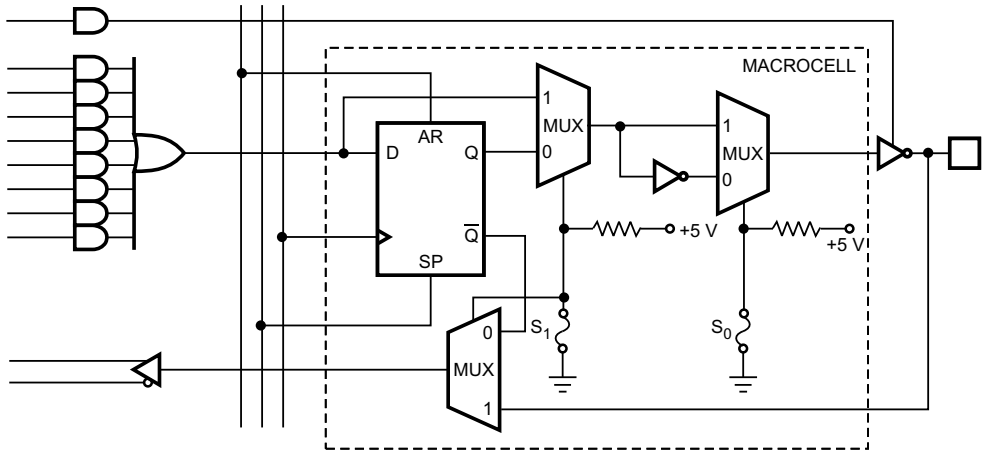
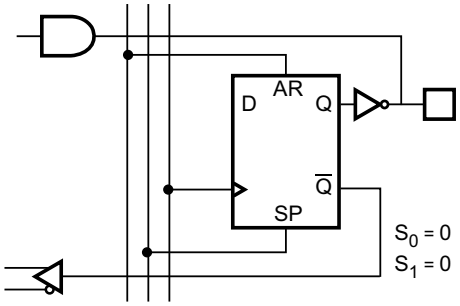
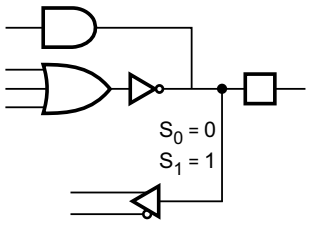


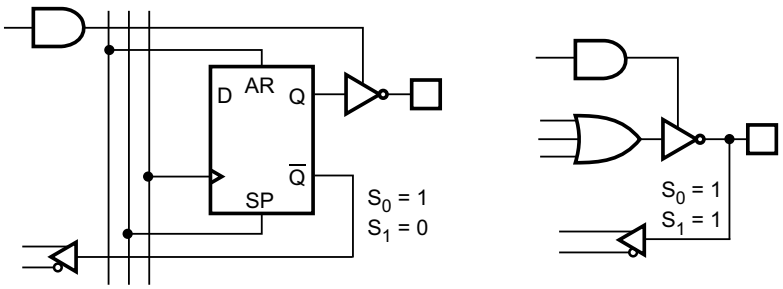
Fig. 10.11.59 Construction of one output macrocell



(a) Registered PAL with inverted output



(b) Combination PAL with inverted output



(c) Registered PAL with non-inverted output **(d) Combinational PAL with non-inverted output**

Fig. 10.11.60 Different ways of configuring PAL output

10.11.7 EX-OR PALs

PALs also are available with EX-OR excitation inputs to the output flip-flop. This allows logic reduction when the flip-flop excitation functions are more easily realized using EX-ORs. Fig. 10.11.61 shows the EX-OR registered output PAL.

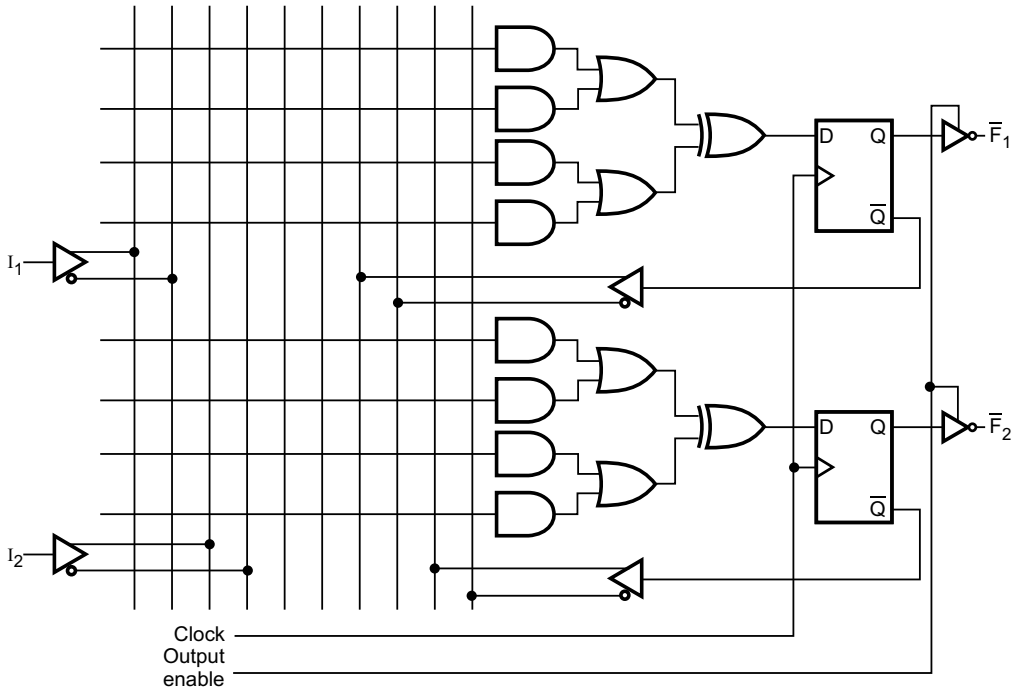


Fig. 10.11.61 Structure of registered PAL with EX-OR excitation input

We have already seen that XOR gates are used to implement fuse configurable output polarity. XORs have other applications as well. Fig. 10.11.61 is the logic diagram for the 20 × 8 device. (Refer Fig. 10.11.62 on next page)

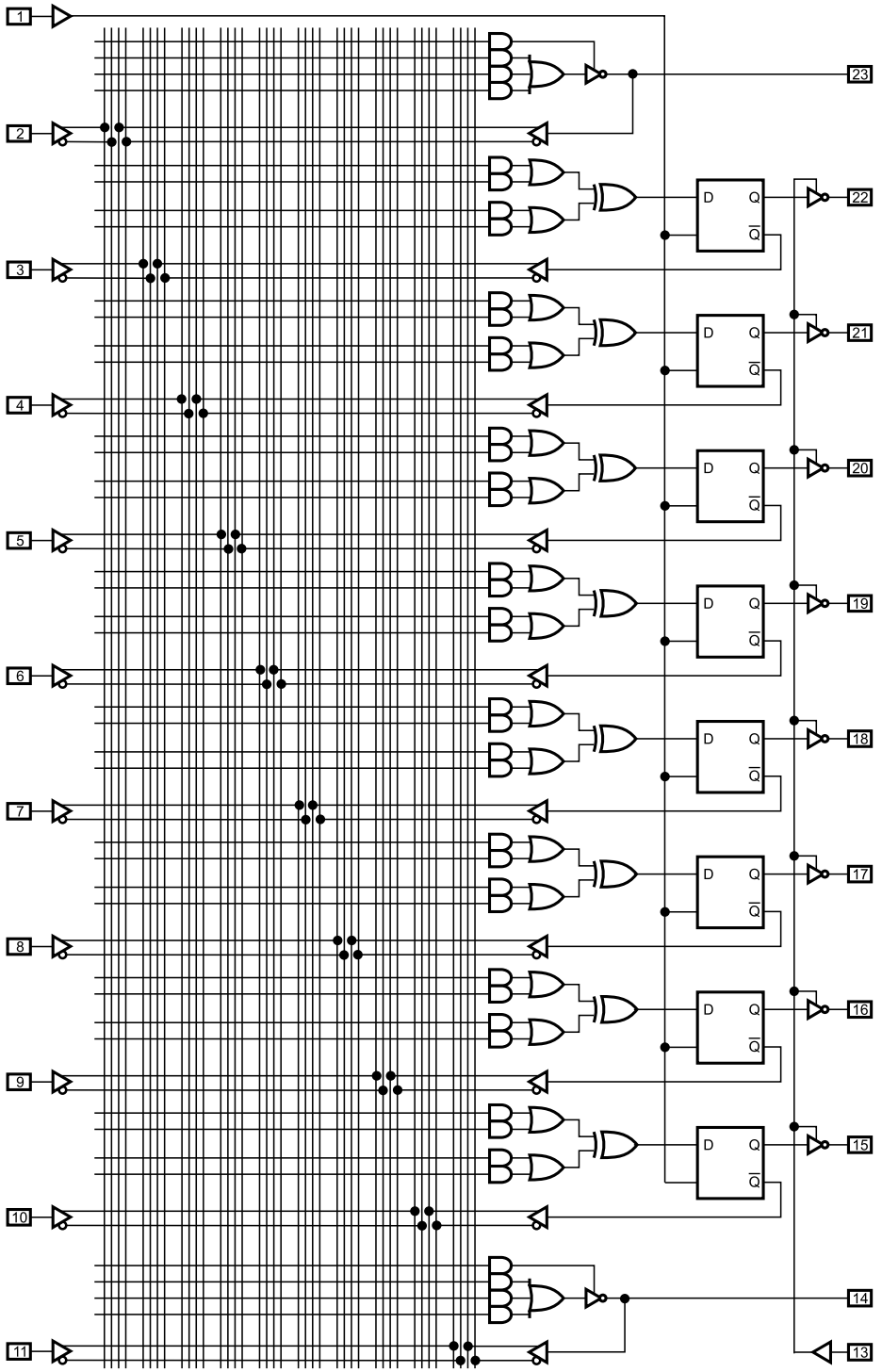


Fig. 10.11.62 Logic diagram for 20 × 8

In the device, flip-flop input is driven by XOR gate. Each of these XOR gates is fed in turn by two sum-of-products arrays of two product terms each. These XORs can be used to reduce the amount of logic required for many applications, particularly counters. The XORs not only allow dynamic polarity control of the outputs, but also allow surprisingly complex designs to be implemented using very few product terms. In recent years, the value of these extra gates has become increasingly apparent to circuit designers.

Review Questions

1. *Describe the concept, working and applications of the PLD.* **May-07, Marks 3**
2. *Discuss on the working of the following programmable logic device : PLD.* **May-09, Marks 2**
3. *Write a note on PLA.* **May-03, 12, Dec.-03, Marks 4**
4. *Discuss the concept, working and applications of PLA.* **Dec.-06, Marks 6**
5. *What is a PLA ?* **May-08, Marks 2**
6. *How does the architecture of a PLA different from a PROM ?* **May-09, Marks 2**
7. *Write a note on PAL.*
8. *Distinguish between a PAL and PLA.* **Dec.-03, Marks 2**
9. *Whether PAL is same as PLA? Explain.* **Dec.-05, May-05, Marks 2**
10. *How does the architecture of a PLA different from a PROM ?* **May-09, Marks 2**
11. *What is PAL ? How does it differ from PLA ?* **Dec.-09, Marks 2**
12. *With logic diagram, explain the basic macrocell.* **May-11, Marks 8**
13. *Write a note on a. Registered PALs b. Configurable PALs c. EX-OR-PALs*

10.12 Field Programmable Gate Arrays (FPGA)

Dec.-04, 08, 10, May-06, 07, 08, 09, 10, 12

Field Programmable Gate Arrays (FPGA) provide the next generation in the programmable logic devices. The word **field** in the name refers to the ability of the gate arrays to be programmed for a specific function by the user instead of by the manufacturer of the device. The word **array** is used to indicate a series of columns and rows of gates that can be programmed by the end user.

As compared to standard gate arrays, the field programmable gate arrays are larger devices. They are more flexible and can be used for more complex implementations of logic circuits that use equivalent of several Millions of logic gates. The basic cell structure for FPGA is some what complicated than the basic cell structure of standard gate array.

The Fig. 10.12.1 shows the general structure of FPGA chip. It consists of a large number of programmable logic blocks called **Configurable Logic Blocks (CLBs)** surrounded by programmable I/O block. The programmable logic blocks of FPGA are

smaller and less capable than a PLD, but an FPGA chip contains a lot more logic blocks to make it more capable. As shown in the Fig. 10.12.1, the logic blocks are distributed across the entire chip. These logic blocks can be interconnected with programmable inter connections.

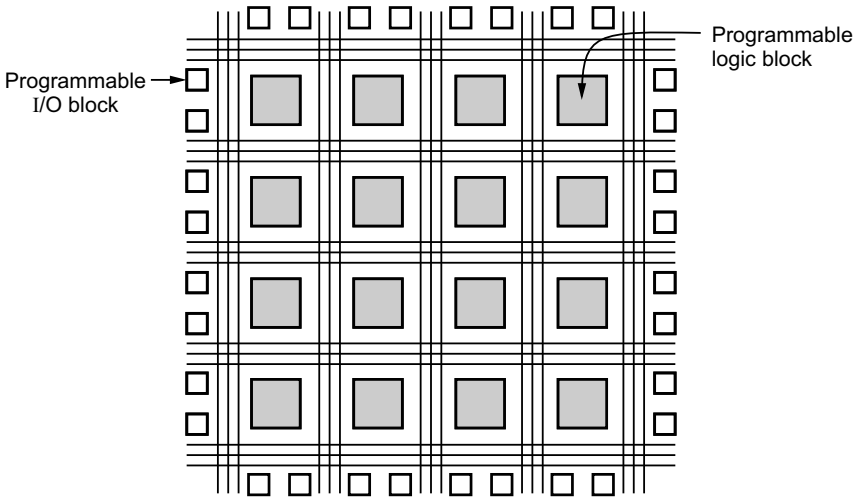


Fig. 10.12.1 General FPGA chip architecture

The FPGA architecture consists of three types of configurable elements :

- A perimeter of input/output blocks (IOBs),
- A core array of configurable logic blocks (CLBs), and
- Resources for interconnection.

The IOBs provide a programmable interface between the internal array of logic blocks (CLBs) and the device's external package pins. CLBs perform user-specified logic functions, and the interconnect resources carry signals among the blocks.

A configuration program stored in internal static memory cells determines the logic functions and the interconnections. The configuration data is loaded into the device during power-up reprogramming functions.

FPGA devices are customized by loading configuration data into internal memory cells. The FPGA device can either actively read its configuration data out of an external serial or byte-wide parallel PROM (master modes), or the configuration data can be written to the FPGA device (slave and peripheral modes).

10.12.1 Applications of FPGA

FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes :

- They are used to build random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more.
- They are used to build higher level embedded functions such as adders and multipliers.
- Some FPGA's have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip.
- FPGAs being parallel processing devices find use in applications like brute force attacks used in breaking cryptographic algorithms, in convolution and FFT computations.
- FPGAs can be used as custom computing machines. This involves using the programmable parts to "execute" software, rather than compiling the software for execution on a regular CPU.
- Other interesting application of FPGAs is prototyping of designs later to be implemented in gate arrays.
- They are also used in medical imaging, reconfigurable computing, speech recognition, cryptography, bioinformatics and so on.

10.12.2 SRAM Based FPGA

May-10

One important class of FPGA is the SRAM-based FPGAs which can be easily reprogrammed any number of times. SRAM-based FPGA stores logic cells configuration data in the static memory (organized as an array of latches). Thus by changing the contents of configuration we can reconfigure FPGA.

A few pins on the chip are dedicated to configuration; some additional pins may be used for configuration and later released for use as general-purpose I/O pins. Because FPGAs are reconfigured relatively infrequently, configuration lines are usually bit-serial. However, it is possible to send several bits in parallel if configuration time is important.

Since SRAM is volatile and can't keep data without power source, such FPGAs must be programmed (configured) upon start. There are two basic modes of programming :

- Master mode, when FPGA reads configuration data from an external source, such as an external Flash memory chip.
- Slave mode, when FPGA is configured by an external master device, such as a processor. This can be usually done via a dedicated configuration interface or via a boundary-scan (JTAG) interface.

SRAM-based FPGA include most chips of Xilinx Virtex and Spartan families and Altera Stratix and Cyclone.

Review Questions

1. *What is FPGA ?* **Dec.-04, May-06, Marks 2**
2. *Describe the concept, working and applications of FPGA.* **Dec.-10, May-07, Marks 6**
3. *Write a note on FPGA.* **May-08, 09, Dec.-08, Marks 2**
4. *Write a note on SRAM based FPGA.* **May-10, Marks 8**
5. *Write a note on FPGA and its application.* **May-12, Marks 8**

10.13 Two Marks Questions and Answers

Q.1 ***Define a memory cell. Give an example.*** **May-07, Dec.-04, 06**

Ans. : Memories are made up of registers. Each register consists of storage elements (flip-flops or capacitors in semiconductor memories and magnetic domain in magnetic storage), each of which stores one bit of data. Such a storage element is called a **memory cell**.

Q.2 ***Define a memory location.*** **May-06**

Ans. : Memories are made up of registers. Each register in the memory is one storage location also called **memory location**. Each memory location is identified by an **address**.

Q.3 ***What is a volatile memory? Give example.*** **Dec.-07**

Ans. : The memory which cannot hold data when power is turned off is known as volatile memory. The static RAM is a volatile memory.

Q.4 ***Name the types of ROM.*** **May-11**

Ans. : There are four types of ROM : Masked ROM, PROM, EPROM and EEPROM or E^2PROM.

Q.5 ***Define address and word.***

Ans. : In a ROM, each bit combination of the input variable is called an address. Each bit combination that comes out of the output lines is called a word.

Q.6 ***Explain ROM.***

Ans. : A Read Only Memory(ROM) is a device that includes both the decoder and the OR gates within a single IC package. It consists of n input lines and m output lines. Each bit combination of the input variables is called an address. Each bit combination that comes out of the output lines is called a word. The number of distinct addresses possible with n input variables is 2^n.

Q.7 ***Define PROM.***

Ans. : PROM is Programmable Read Only Memory. It consists of a set of fixed AND gates connected to a decoder and a programmable OR array.

Q.8 *What is mask-programmable?*

Ans. : With a mask programmable PLA, the user must submit a PLA program table to the manufacturer.

Q.9 *Define PLD.*

Ans. : Programmable Logic Devices consist of a large array of AND gates and OR gates that can be programmed to achieve specific logic functions.

Q.10 *Give the classification of PLDs.*

Ans. : PLDs are classified as :

- PROM(Programmable Read Only Memory),
- Programmable Logic Array(PLA),
- Programmable Array Logic (PAL), and
- Generic Array Logic(GAL)

Q.11 *What is a PLA ?* **May-08**

Ans. : **PLA** stands for **Programmable Logic Array**, which is a LSI component. In PLA, both AND and OR gates have fuses at the inputs, therefore in PLA both AND and OR gates are programmable. The outputs from OR gates go through fuses as inputs to output inverters so that final output can be programmed as either AND-OR or AND-OR-INVERT.

Q.12 *Whether PAL is same as PLA? Explain.* **Dec.-03, 05, May-05**

Ans. : Refer section 10.11.4.

Q.13 *What is the advantage of PLA over ROM ?* **May-10**

Ans. : Refer section 10.11.4.

Q.14 *How does the architecture of a PLA different from a PROM ?* **May-09**

Ans. : Refer section 10.11.4.

Q.15 *Why was PAL developed ?*

Ans. : PAL is a PLD that was developed to overcome certain disadvantages of PLA, such as longer delays due to additional fusible links that result from using two programmable arrays and more circuit complexity.

Q.16 *Why the input variables to a PAL are buffered?*

Ans. : The input variables to a PAL are buffered to prevent loading by the large number of AND gate inputs to which available or its complement can be connected.

Q.17 *What is FPGA?* **Dec.-04, May-06**

Ans. : **FPGA** stands for **field programmable gate array**, which is the next generation in the programmable logic devices. The word **field** refers to the ability of the gate arrays to be programmed for a specific function by the end user. The word **array** indicates a series of columns and rows of gates that can be programmed by the end user.

Q.18 *List the configurable elements in the FPGA architecture.*

Ans. : The FPGA architecture consists of three types of configurable elements :

1. A perimeter of Input/Output Blocks (IOBs).
2. A core array of Configurable Logic Blocks (CLBs).
3. Resources for interconnection.

Q.19 *How is individual location in a EEPROM programmed or erased ?* **May-06**

Ans. : Since it is electrically erasable memory, by activating particular row and column it is possible that individual can be programmed or erased.

Q.20 *Explain static memory.* **Dec.-06, May-07**

Ans. : Refer section 10.5.1.

Q.21 *Mention the two types of erasable PROM.* **Dec.-06**

Ans. : Refer sections 10.3.2 and 10.3.3.

Q.22 *Whether ROM is classified as nonvolatile storage device? Why?* **May-07**

Ans. : Refer section 10.3.

Q.23 *Write the advantages of EPROM over PROM.* **May-07**

Ans. : Refer section 10.3.2.

Q.24 *What is write cycle time ?* **May-08**

Ans. : Refer section 10.8.

Q.25 *Explain 'write operation' with an example.* **Dec.-08**

Ans. : Refer section 10.1.

Q.26 *What is memory cycle ?* **May-09**

Ans. : Refer section 10.8.

Q.27 *Compare the features of PROM, PAL and PLA.* **May-09**

Ans. : Refer section 10.11.4.

Q.28 *Compare and contrast static RAM and dynamic RAM.* **Dec.-09**

Ans. : Refer section 10.5.2.2.

Q.29 *What is PAL ? How does it differ from PLA ?* **Dec.-09**

Ans. : Refer sections 10.11.3 and 10.11.4.

Q.30 *What is meant by memory expansion? Mention its limit.* **May-10**

Ans. : Refer section 10.10.

Q.31 *What are the advantages of static RAM compared to Dynamic RAM ?* **May-10**

Ans. : Refer section 10.5.2.2.

Q.32 *What is access time and cycle time of a memory ?* **Dec.-10**

Ans. : Refer section 10.8.

Q.33 *What is a combinational PLD?* **Dec.-07**

Ans. : The PROM is a combinational programmable logic device (combinational PLD). A combinational PLD is an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND-OR sum of product implementation. There are three major types of combinational PLDs and they differ in the placement of the programmable connections in the AND-OR array. The types of combinational PLDs are :

- PROM : Fixed AND array and programmable OR array
- PAL : Programmable AND array and a fixed OR array
- PAL : Programmable AND array and a fixed OR array

Q.34 *Mention few applications of PLA and PAL.*

Ans. : Applications of PLA and PAL are :

- They are used to implement combinational circuits.
- Registers PALs can be used to implement sequential circuits.
- They are used as a combinational logic in addressing decoding section of a microprocessor based systems.
- They can be programmed to use as code converters.

❑❑❑

11 Design of Combinational and Sequential Circuits using VERILOG

Syllabus

Design of combinational and sequential circuits using VERILOG.

Contents

11.1 Introduction

11.2 Structure of Verilog Module

11.3 Operators in Verilog HDL

11.4 Verilog Data Types

*11.5 HDL for Combinational Circuits . . . . . **Dec.-10, 12,** · · · · · · · · · · · · · Marks 2*

11.6 HDL for Sequential Circuits

11.1 Introduction

We are familiar with the design of a digital system. The basic steps involved in this process are,

a. Specify the desired behaviour of the circuit.

b. Synthesize the circuit.

c. Implement the circuit.

d. Test the circuit to check whether the desired specifications meet.

But as the size and complexity of digital systems increase, they can not be designed manually; their design becomes highly complex. At their most detailed level, they may consists of millions of elements, i.e. transistors or logic gates. So Computer Aided Design (CAD) tools are used in the design of such systems. One such a tool is a **Hardware Description Language (HDL)**. The most prominent modern HDLs in industry are Verilog and VHDL. Verilog is one of the two major Hardware Description Languages (HDLs) used by hardware designers in industry and academia.

Review Question

1. *Explain the need of HDL.*

11.2 Structure of Verilog Module

The Verilog HDL describes a digital system as a set of modules. Each of these modules has an interface to other modules to describe how they are interconnected. Each module consists of a **declaration** and a **body**. In the declaration, name, inputs and outputs of the module are listed. The body shows the relationship between the inputs and the outputs. Usually, we place one module per file but that is not a requirement. The modules may run concurrently, but usually we have one top level module which specifies a closed system containing both test data and hardware models.

A module is a basic building block of Verilog HDL. Modules can represent pieces of hardware ranging from simple gate to complete systems. e.g. a microprocessor. The structure of module is,

```
module <module name> <port list>;
<declares>
<module items>
 endmodule
```

The **<module name>** is an identifier that uniquely names the module. The module name is user selected. It should start with alphabetical letter and it can include the special character underscore (_). In contrast to VHDL, Verilog is a case sensitive. The

<port list> is a list of input, inout and output ports which are used to connect to other modules. The **<declares>** section specifies data objects as registers, memories and wires as well as procedural constructs such as **functions and tasks.**

The Listing 12.2.1 shows the example Verilog code. This code is the description for the logic circuit shown in the Fig. 11.2.1.

```
module BG_circuit (P, Q, R, Y);
input P, Q, R;
output Y;
wire s, t;
assign s = ~P;
assign t = s & Q;
assign Y = t | R;
endmodule
```

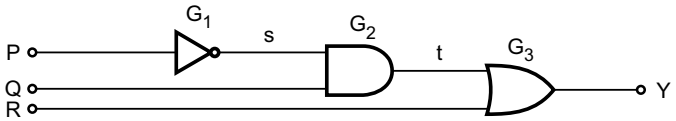


Fig. 11.2.1 Circuit using basic gates

Important points to remember while representing any module using Verilog HDL.

1. Each statement in Verilog HDL except comment and last statement (endmodule) is terminated with a semicolon (;).
2. The blank lines are allowed in the module and also spaces between the words or at the beginning of the line are allowed.
3. The language is case sensitive. i.e. the uppercase and lowercase letters are considered as different.
4. The function of a circuit is indicated by the text between two slashes (//) and the end of the line which is interpreted as a comment.
5. Verilog uses about 100 keywords. All must be given in lowercase.
6. Identifiers are the names given to variables. With these names, they can be referred in the design. They consist of alphanumeric characters and underscore (–). They can not start with a number.
7. The **input** and **output** keywords are used for declaring inputs and outputs. The keyword **inout** is used for a signal that is both, an input and an output.
8. Internal connections within the circuit are declared with the keyword **wire**.

Note : The keywords are highlighted by printing them in bold. But it is not the requirement of Verilog HDL.

Review Question

1. *Explain the structure of Verilog HDL program with the help of example.*

11.3 Operators in Verilog HDL

Verilog HDL includes following kinds of operators :

- Boolean logical
- Unary reduction logical
- Bitwise logical
- Relational
- Binary arithmetic
- Unary arithmetic
- Other

Boolean logical operators

Logical operators operate on logical operands and return a logical value, i.e., TRUE(1) or FALSE(0). Used typically in **if** and **while** statements. Do not confuse logical operators with the bitwise Boolean operators. For example, ! is a logical NOT and ~ is a bitwise NOT. The first negates, e.g. !(5 == 6) is **TRUE.** The second complements the bits,

e.g. ~{1,0,1,1} is 0100.

Operator	Name
!	Logical negation
&&	Logical AND
| |	Logical OR

Unary reduction logical operators

Unary reduction operators operate on a single operand. They produce a single bit result from applying the operator to all of the bits of the operand. For example, in statement **B = &A**, if A = 1101, then B = (1 & 1 & 0 & 1) = 0.

Operator	Name
&	AND reduction
|	OR reduction
^	XOR reduction
~&	NAND reduction
~|	NOR reduction
~^	XNOR reduction

Bitwise logical operators

Bitwise operators operate on the bits of the operand or operands. The result of A & B is the AND of each corresponding bit of A with B. For example, if A = 1011 and B = 0101, then C = A & B gives C = 0001. Except for bitwise negation, these operators operate on a two operands.

Operator	Operation
~	Bitwise negation
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~&	Bitwise NAND
~ |	Bitwise NOR
~^ or ^~	Equivalence bitwise NOT XOR

Relational operators

Relational operators compare two operands and return a logical value, i.e. TRUE(1) or FALSE(0). For example, if A = 0100 and B = 0100, then statement if (A == B) results True(1). If any bit is unknown, the relation is ambiguous and the result is unknown(X).

Operator	Operation
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Logical equality
!=	Logical inequality

Binary arithmetic operators

Binary arithmetic operators operate on two operands. Register and net, i.e. wire, operands are treated as unsigned. However, real and integer operands may be signed. If any bit of an operand is unknown ('x') then the result is unknown.

Operator	Operation	Comments
+	Addition	
–	Subtraction	

*	Multiplication	
/	Division	Divide by zero produces an x, i.e. unknown.
%	Modulus	

Unary arithmetic operators

Operator	Operation	Comments
–	Unary minus	Changes sign of its operand.

Other operators

The conditional operator operates much like in the language C.

Operator	Operation	Comments
===	Case equality	The bitwise comparison includes comparison of x and z values. All bits must match for equality. Returns TRUE or FALSE.
!==	Case inequality	The bitwise comparison includes comparison of x and z values. Any bit difference produces inequality. Returns TRUE or FALSE.
{ , }		Concatenation joins bits together with 2 or more comma-separated expressions, e.g. {A[0], B[1:7]} concatenates the zero bit of A to bits 1 to 7 of B.
<<	Shift left	Vacated bit positions are filled with zeros, e.g. A = A < 2; //shifts A two bits to left with zero fill.
>>	Shift right	Vacated bit positions are filled with zeros.
?:	Conditional	Assigns one of two values depending on the conditional expression e.g. A = C>D ? B+3 : B–2 means if C greater than D, the value of A is B+3 otherwise B – 2.

Operator precedence

The precedence of operators is shown in Table 11.3.1. The top of the table is the highest precedence and the bottom is the lowest. Operators on the same line have the same precedence and associate left to right in an expression. Parentheses can be used to change the precedence or clarify the situation. We strongly urge you to use parentheses to improve readability.

Type	Operators
Unary operators	! & ~& | ~| ^ ~^ + – (Highest precedence)
Multiplying operators	* / %
Sign operators	+ –

Relational operators	<< >> < <= > >= == != === ~==
Logical operators	& ~& ^ ~^ | ~ | && ||
Conditional operators	?: (Lowest precedence)

Table 11.3.1 Operator precedence in Verilog HDL

Review Question

1. *If A = 1011 and B = 1101, find the value of the following expressions :*
   a. A & B *b. A && B*
   c. A ^ B *d. A ~^ B* *e. A >> 2*

11.4 Verilog Data Types

The set of Verilog HDL data types is designed to represent the data storage and transmission elements found in digital hardware. Since the purpose of Verilog HDL is to model digital hardware, the primary data types are for modeling registers **(reg)** and nets (**wire**). The **reg** variables store the last value that was procedurally assigned to them whereas the **wire** variables represent physical connections between structural entities such as gates. A **wire** does not store a value; its value changes continuously by the circuit that are driving it.

The Verilog HDL also supports several other data types including integers, real, parameters and arrays.

Nets (wire) and registers

The **reg** and **wire** data objects may have the following possible values :

Value	Definition
0	Logical zero or false
1	Logical one or true
x	Unknown logical value
z	High impedance of tristate gate

The **reg** variables are initialized to **x** (unknown logic value) at the start of the simulation. Any **wire** variable not connected to something has the x value. We may specify the size of a register or wire in the declaration. For example, the declarations

```
wire D1;
wire D0 = 1'b0;
reg flag;
```

Specify wires **D1** and **D0** to be single bit wide. The initial value of D0 is 1'b0, which represents 1 bit with value 0.

When the size of the **reg** or **wire** is more than 1 bit then registers and wires are declared as **vectors**. Vectors are declared by brackets. The bits in vectors can be referenced by the notation **[<start-bit>:<end-bit>]**. For example, the declarations

```
reg [0:7] A, B;
wire [0:3] Dataout;
reg [7:0] C = 8'b10001010;
reg [7:0] D = 8'd138;
A = 8'b01011010
B = {A[0:3] ¦ A[4:7], 4'b0000};
```

Specify registers **A** and **B** to be 8-bit wide with the most significant bit the zeroth bit, whereas the most significant bit of register **C** and register **D** is bit seven. The wire **Dataout** is 4 bits wide. **C** holds a value of 10001010 (b stands for binary). **D** holds the same value as **C** (10001010); however it is specified in decimal 138 (d stands for decimal).

B is set to the first four bits of **A** bitwise or-ed with the last four bits of **A** and then concatenated with 0000. **B** now holds a value of 11110000. The {} brackets means the bits of the two or more arguments separated by commas are concatenated together.

Review Question

1. *Explain various data types supported by Verilog.*

11.5 HDL for Combinational Circuits

Dec.-10, 12

There are three modeling techniques which can be used for describing a module :

- Gate-level modeling / Structural modeling
- Dataflow modeling
- Behavioral modeling

A module can be described in any one of the above techniques or by taking combination of them.

11.5.1 Structure of the Data Flow Description

Data flow describes how the circuit signals flow from the inputs to the outputs. There are some concurrent statements which allow to describe the circuit in terms of operations on signals and flow of signals in the circuit. When such concurrent statements are used in a program, the style is called a **'dataflow design'**. Concurrent signal assignment statements are used in this type of modeling style.

In Verilog, predefined word **assign** is used to assign a value to the left-hand side of a signal-assignment statement.

Example 11.5.1 *Design a 3 input AND gate using verilog.* **Dec.-12, Marks 2**

Solution : module AND_3 (A, B, C, Y);

```
    input    A, B, C;
    output   Y;
    assign   Y = A & B & C;
    end module
```

Example 11.5.2 *Verilog HDL code for full adder in data flow description.*

Solution :

```
module full_add (A, B, Cin, Cout, Sum);
input A, B, Cin;
output Sum, Cout;
    assign Sum = (A ^ B) ^ Cin;
    assign Cout = (A & B) | (Cin & A) | (Cin & B);
endmodule
```

The built in operators of VHDL (for example : AND, OR, NOT) and Verilog (for example & | ^) are used in the expression.

Example 11.5.3 *Verilog HDL code for given combinational circuit.*

Solution : The listing shows the HDL code for the circuit shown in the Fig. 11.5.1. The circuit is an AND-OR circuit in which signals A, B, C, D and E are input signals, signal Y is an output signal and signals I1 and I2 are intermediate signals.

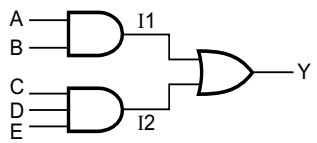


Fig. 11.5.1 AND-OR circuit

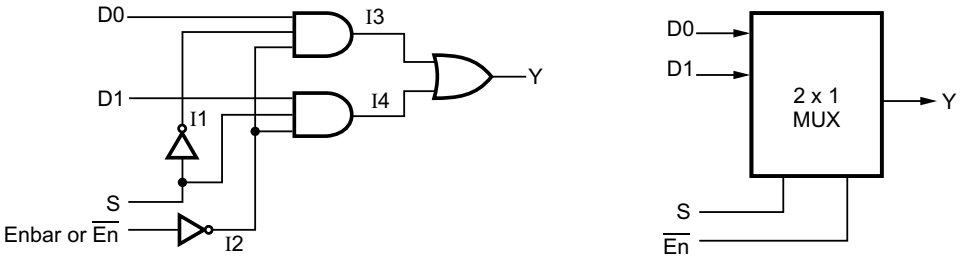
Verilog AND-OR Circuit Description.

```
module AND_OR (A, B, C, D, E, Y);
input A, B, C, D, E;
output Y;
wire I1, I2;
assign #10 I1 = A & B;
assign #10 I2 = C & D & E;
```

```
assign #10 Y = I1 ¦ I2;
endmodule
```

Example 11.5.4 *Verilog HDL code for multiplexer with active low enable.*

Solution : The Fig. 11.5.2 shows 2×1 multiplexer. It has two 1-bit inputs : D0 and D1, a 1-bit select line S, a 1-bit output : Y and a 1-bit enable signal : $\overline{En}$. The input signals D0 and D1 are connected to the one input of AND gates : AND1 and AND2, respectively. If enable signal ($\overline{En}$) is low, the output is equal to one of the two inputs depending on the status of select (S) signal. If S = 0, the output is equal to D0 and if S = 1 the output is equal to D1. The Table 11.5.1 shows the truth table for 2×1 multiplexer.



(a) Logic diagram (b) Logic symbol

Fig. 11.5.2 2 × 1 multiplexer

Input		Output
S	**Enbar**	**Y**
X	1	0
0	0	D0
1	0	D1

Table 11.5.1 Truth table for a 2×1 multiplexer

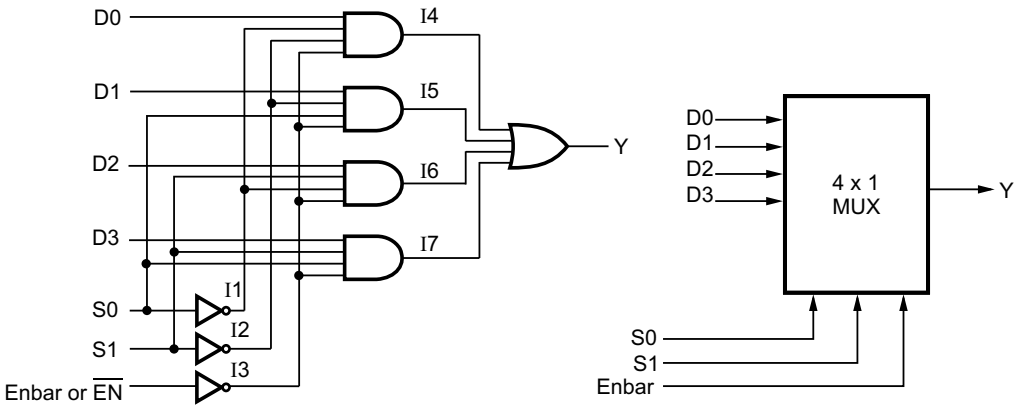
```
module mux 2 × 1 (D0, D1, S, Enbar, Y);
input D0, D1, S, Enbar;
output Y;
wire I1, I2, I3, I4;
                    //    Assume 10 time units delay for all and, or, not. In Verilog
                    //    we cannot use specific time units, such as nanoseconds.
                    //    The delay here is expressed in simulation screen units.

    assign #10 Y = I3 ¦ I4;
    assign #10 I3 = D0 & I1 & I2;
```

```
    assign #10 I4 = D1 & S & I2;
    assign #10 I1 = ~ S;
    assign #10 I2 = ~ Enbar;
endmodule
```

Example 11.5.5 *Verilog HDL code for* 4×1 *multiplexer.* **Dec.-12, Marks 8**

Solution : The Fig. 11.5.3 shows 4×1 multiplexer. Each of the four lines, D0 to D3, is applied to one input of an AND gate. Selection lines are decoded to select a particular AND gate.



(a) Logic diagram **(b) Logic symbol**

Fig. 11.5.3 4 × 1 multiplexer

Input			Output
S1	S0	Enbar	Y
X	X	1	0
0	0	0	D0
0	1	0	D1
1	0	0	D2
1	1	0	D3

Table 11.5.2 Truth table for 4 × 1 multiplexer

```
module mux 4 × 1 (D, S, Enbar, Y);
input [3 : 0] D;
input S, Enbar;
output Y;
wire I1, I2, I3, I4, I5, I6, I7;
                    //    Assume 10 time units delay for all and, or, not. In
                    //    Verilog we cannot use specific time units, such as
```

```
                    //      nanoseconds. The delay here is expressed in simulation
                    //      screen units.
    assign #10 Y = I4 ¦ I5 ¦ I6 ¦ I7;
    assign #10 I4 = D0 & I1 & I2 & I3;
    assign #10 I5 = D1 & S0 & I2 & I3;
    assign #10 I6 = D2 & S1 & I1 & I3;
    assign #10 I7 = D3 & S0 & S1 & I3;
    assign #10 I1 = ~ S0;
    assign #10 I2 = ~ S1;
    assign #10 I3 = ~ Enbar;
endmodule
```

Example 11.5.6 *Verilog HDL code for 2 × 2 unsigned combinational array multiplier.*

Solution : Let us generalize the multiplication process for a 2 × 2 multiplier for two unsigned 2-bit numbers : multiplicand A = A1 A0 and multiplier B = B1 B0. The Fig. 11.5.4 shows how the multiplication process is carried out.

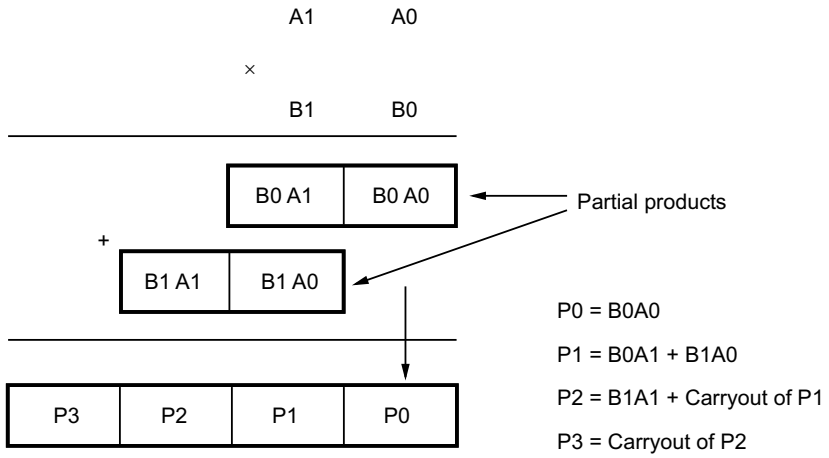


Fig. 11.5.4 Multiplication process

The multiplication process involves multiplication (product) of 2-bit number and addition of 2-bit number. The multiplication of 2-bits can be implemented using 2-input AND gate whereas addition of 2-bits can be implemented using half-adder. Such an implementation of 2 × 2 multiplier is shown in the Fig. 11.5.5.

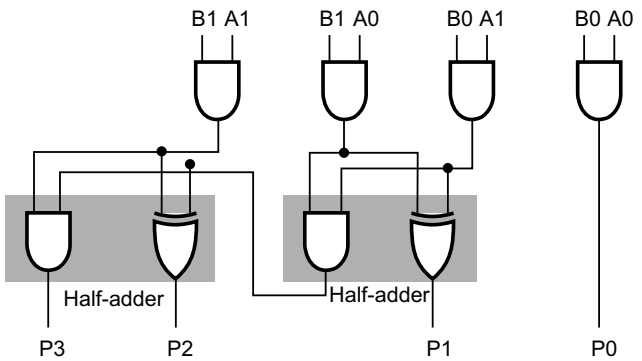


Fig. 11.5.5 2 × 2 bit combinational array multiplier

```
module Arr_Mul (A, B, P);
input [1:0] A, B;
output [3:0] P;
                                    /*   For simplicity, propagation delay times are not
                                         considered in this example.*/
      assign P[0] = B[0] & A[0];
      assign P[1] = (B[0] & A[1]) ^ (B[1] & A[0]);
      assign P[2] = (B[1] & A[1]) ^ ((B[0] & A[1]) & (B[1] & A[0]));
      assign P[3] = (A[1] & B[1]) & ((B[0] & A[1])& (B[1] & A[0]));
 endmodule
```

Example 11.5.7 *Verilog HDL code for 2 × 2 magnitude comparator.*

Solution :

```
module compr_2 (A, B, AgtB, AltB, AeqB);
input [1:0] A, B;
output AgtB, AltB, AeqB;
      assign AgtB = (A[0] & ~B[1] & ~ B[0]) ¦ (A[1] & ~ B[1])
                          ¦ (A[1] & A[0] & ~B[0]);

      assign AltB =     (~A[1] & ~ A[0] & B[0]) ¦ (~ A[0] & B[1] & B[0]) ¦
                          (~ A[1] & B[1]]);
      assign AeqB = (A[0]  B[0]) & (A[1] B[1]);
endmodule
```

Example 11.5.8 *Verilog HDL code for 4-bit ripple-carry and carry-lookahead adder.*

Solution : A single full-adder is capable of adding two one-bit numbers and an input carry. In order to add binary numbers with more than one bit, additional full-adders can be employed. A 4-bit, parallel adder can be constructed using number of full adder circuits connected in parallel. Fig. 11.5.6 shows the block diagram of 4-bit parallel adder using number of full-adder circuits connected in cascade, i.e. the carry output of each adder is connected to the carry input of the next higher-order adder.

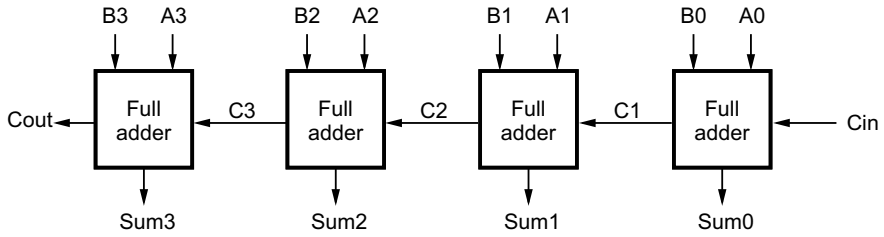


Fig. 11.5.6 4-bit parallel adder

It should be noted that either a half-adder can be used for the least significant position or the carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

```
module RCarry_adder (A, B, Cin, Sum, Cout);
input [3:0] A, B;
input Cin;
output [3:0] sum;
output Cout;
wire C1, C2,C3;
time delay_gt = 7;
                                    //    Assume 7.0 ns propagation delay for all gates.

    assign #(2*delay_gt) sum[0] = (B[0] ^ A[0]) ^ Cin;
    assign #(2*delay_gt) sum[1] = (B[1] ^ A[1]) ^ C1;
    assign #(2*delay_gt) sum[2] = (B[2] ^ A[2]) ^ C2;
    assign #(2*delay_gt) sum[3] = (B[3] ^ A[3]) ^ C3;
    assign #(2*delay_gt) C1 = (A[0] & B[0]) ¦ (A[0] & Cin) ¦ (B[0] & Cin);
    assign #(2*delay_gt) C2 = (A[1] & B[1]) ¦ (A[1] & C1) ¦ (B[1] & C1);
    assign #(2*delay_gt) C3 = (A[2] & B[2]) ¦ (A[2] & C2) ¦ (B[2] & C2);
    assign #(2*delay_gt) Cout = (A[3] & B[3]) ¦ (A[3] & C3) ¦ (B[3] & C3);
endmodule
```

11.5.2 Structure of the Behavioral Description

It is sometimes possible to directly describe the behavior or the functionality of a circuit. Such a modeling style is called behavioral modeling which is very similar in syntax and semantics to that of a high-level programming language (For example : C, Pascal). A behavioral description models the system as to how the outputs behave with the inputs.

In Verilog, the key mechanism used to model the behavior is predefined words **always** or **initial.**

Example 11.5.9 *Verilog HDL code for full adder in behavioral description.*

Solution :

```
module full_add (A, B, Cin, Cout, Sum);
input A, B, Cin;
output Sum, Cout;
reg  Sum, Cout;
always @(A, B, Cin)
        begin
```

```
        Sum = (A ^ B) ^ Cin;
        Cout = (A & B) ¦ (Cin & A) ¦ (Cin & B);
        end
endmodule
```

Example 11.5.10 *Verilog HDL code for half-adder in behavioral description.*

Solution :

```
module half_adder (A, B, Sum, Cout);
input A;
input B;
output Sum;
output Cout;
reg Sum, Cout;
                        /*  Since Sum and Cout are outputs and they are written
                            inside "always," they should be declared as reg */
always @(A, B)
begin
        #10 Sum = A ^ B ;   //   ^ is a bitwise xor logical operator.
        #10 Cout = A & B ;  //   & is a bitwise logical "and" operator
                            /*  The above two statements are procedural
                            (inside always) signal-assignment statements with
                            10 simulation screen units delay*/
end
endmodule
```

Example 11.5.11 *Verilog HDL code in behavioral description of half subtractor.*

Solution :

```
module half_sub (A, B, Diff, Borrow);
input A, B ;
output Diff, Borrow;
reg Diff, Borrow
always @ (A, B)
        begin
                Diff = A ^ B ;
                Borrow = (~ A) & B ;
        end
endmodule
```

Example 11.5.12 *Verilog HDL code in behavioral description of full subtractor.*

Solution :

```
module full_sub (A, B, Bin, Diff, Bout) ;
input A, B, Bin;
output Diff, Bout ;
reg Diff, Bout
always @ (A, B, Bin)
    begin
        Diff = A ^ B ^ Bin ;
        Bout= (~ A & B)   (~ A & Bin) (B  ^ Bin);
    end
endmodule
```

Example 11.5.13 *Verilog HDL code in behavioral description of 2 × 1 multiplexer using IF ELSE.*

Solution :

```
module mux 2 × 1 (D0, D1, S, Enbar, Y);
input D0, D1, S, Enbar;
output Y;
reg Y;
always @ (S, D0, D1, Enbar)
    begin
        if (Enbar == 1)
            Y = 1'bz;
        else
        begin
            if (S)
                Y = D1;
                    /*  This is a procedural assignment. Procedural assignments
                        are used to assign values to variables declared as regs
                        (as Y here in this module). Procedural statements have to
                        appear inside always, blocks, initial, tasks, or functions */
        else
            Y = D0;
            end
    end
endmodule
```

Example 11.5.14 *Verilog HDL code in behavioral description of a 2 × 1 multiplexer using ELSE IF.*

Solution :

```
module mux2×1 (D0, D1, S, Enbar, Y);
input D0, D1, S, Enbar;
output Y;
reg Y;
always @ (S, D0, D1, Enbar)
      begin
            if (Enbar ==0 & S == 1)
            begin
                  Y =  D0;
            end
                  else if (Enbar ==0 & S == 0)
                  Y =  D1;
            else
                  Y = 1'bz;     // b represents binary and z represents high impedance state
            end
endmodule
```

Example 11.5.15 *Verilog HDL code in behavioral description of a priority encoder using casex*

Solution : A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

Table 11.5.3 shows truth table of 4-bit priority encoder.

Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Table 11.5.3 Truth table of 4-bit priority encoder

Table 11.5.3 shows D_3 input with highest priority and D_0 input with lowest priority. When D_3 input is high, regardless of other inputs output is 11. The D_2 has the next priority. Thus, when D_3 = 0 and D_2 = 1, regardless of other two lower priority input, output is 10. The output for D_1 is generated only if higher priority inputs are 0, and so

on. The output V (a valid output indicator) indicates, one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs (Y_1 and Y_0) of the circuit are not used.

```
module Encoder (D, Y);
input [0:3] D;
output [1:0] Y;
reg [1:0] Y;
always @ (D)
      begin
            casex (D)
                  4'b1000 : Y = 2'b00; V = 1'b1
                  4'bx100 : Y = 2'b01; V = 1'b1
                  4'bxx10 : Y = 2'b10; V = 1'b1
                  4'bxxx1 : Y = 2'b11; V = 1'b1
                  default : Y =  2'bzz; V = 1'b0          // D0-D3 = 0000
            endcase
            end
endmodule
```

11.5.3 Gate Level/Structural Description

In structural design, Verilog uses components or gates to model the system. The listing shows the structural description of half-adder.

```
module system (a, b, Sum, Cout);
input a, b;
output Sum, Cout;
            xor X1 (Sum, A, B);       //    X1 is an identifier. It is optional it can be omitted.
            and A1 (Cout, a, b);      //    A1 is optional identifier; it can be omitted.
endmodule
```

Verilog recognizes logic gates. The Fig. 11.5.7 shows all the gates recognized by the Verilog. Like VHDL, the statements in the structural Verilog are concurrent; they are event driven and their order of appearance in the module is irrelevant.

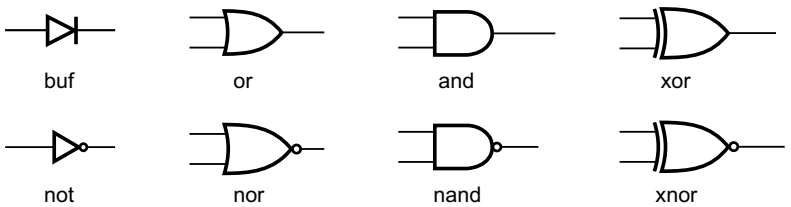
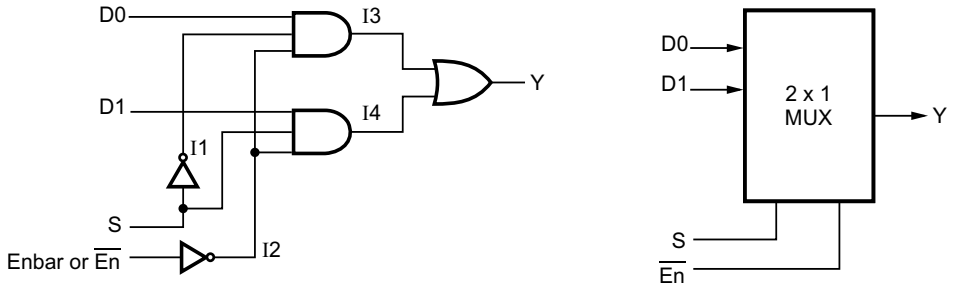


Fig. 11.5.7 Built-in gates in Verilog

Example 11.5.16 *Verilog HDL code in structural description of a 2×1 multiplexer with active low enable.*

Solution : We have seen the behavioral description of 2×1 multiplexer in example 11.5.2. The same 2×1 multiplexer is redrawn here for convenience.



(a) Logic diagram **(b) Logic symbol**

Fig. 11.5.8 2 × 1 multiplexer

```
module mux2 × 1 (D0, D1, S, Enbar, Y);
input D0, D1, S, Enbar;
output Y;
      and #7 (I3, D0, I1, I2);
      or #7 (Y, I3, I4);
      and #7 (I4, D1, S, I2);
      not #7 (I1, S);
      not #7 (I2, Enbar);
endmodule
```

Example 11.5.17 *Verilog HDL code in structural description of a 2×4 decoder with enable input.*

Solution : The Fig. 11.5.9 shows the logic circuit for 2 to 4 line decoder.

```
module decoder2 × 4 (A, B, En, Y);
input A, B;
input En;
output [3:0] Y;
wire Abar, Bbar;
      not (Abar, A);
      not (Bbar, B);
      and (Y0, Abar, Bbar, En);
      and (Y1, Abar, B, En);
      and (Y2, A, Bbar, En);
      and (Y3, A, B, En);
Endmodule
```

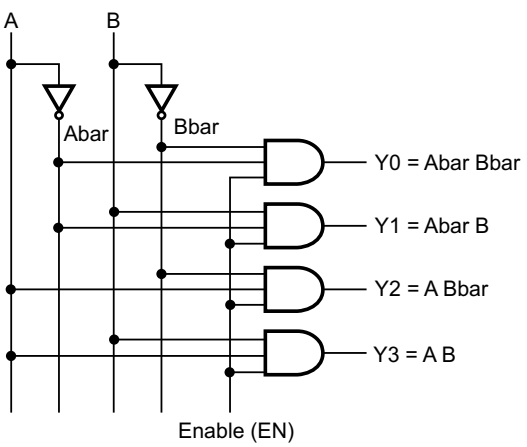


Fig. 11.5.9 2-to-4 line decoder

Example 11.5.18 *Verilog HDL code in structural description of a 2×4 decoder with tri-state output.*

Solution : The Fig. 11.5.10 shows the 2×4 decoder with tri-state output. For this decoder when enable (En) input is low, the outputs are in high impedance state, i.e. tri-state.

The Table 11.5.4 shows the truth table for a 2×4 decoder with tri-state output.

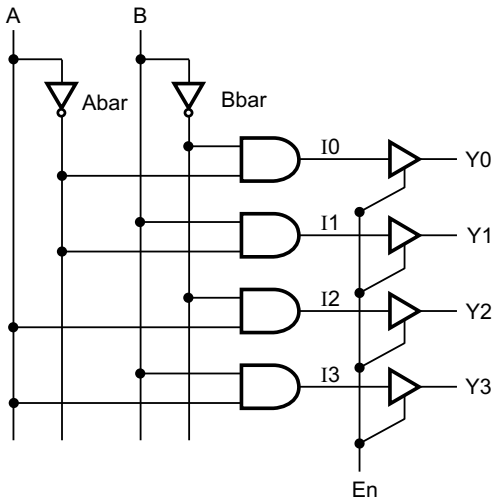


Fig. 11.5.10 2×4 decoder with tri-state output

Inputs			Outputs			
En	**A**	**B**	**Y3**	**Y2**	**Y1**	**Y0**
0	X	X	Z	Z	Z	Z
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Table 11.5.4 Truth table for 2×4 decoder with tri-state output

Verilog has built-in buffers. The Fig. 11.5.11 shows the buffers supported by Verilog.

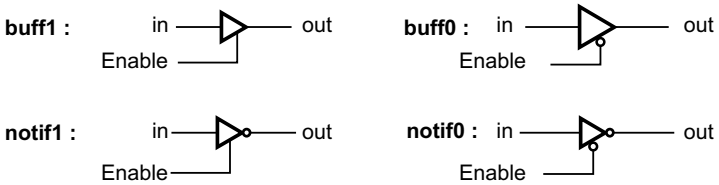


Fig. 11.5.11 Built-in buffers in Verilog

```
module decoder2 × 4 (A, B, En, Y);
input A, B;
input En;
output [3:0] Y;
wire Abar, Bbar;
        bufif1 (Y[0], I0, En);
        bufif1 (Y[1], I1, En);
        bufif1 (Y[2], I2, En);
        bufif1 (Y[3], I3, En);
```

```
    not (Abar, A);
    not (Bbar, B);
    and (I0, Abar, Bbar);
    and (I1, Abar, B);
    and (I2, A, Bbar);
    and (I3, A, B);
endmodule
```

Example 11.5.19 *Verilog HDL code in structural description of a full-adder.*

Solution : The Fig. 11.5.12 shows the implementation of a full-adder with two half-adders and OR gate. The listing given below shows the HDL code for this full-adder.

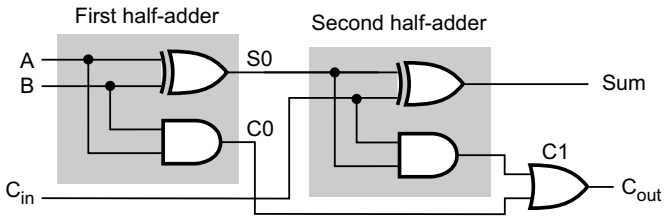


Fig. 11.5.12 Implementation of a full-adder with two half-adders and an OR gate

```
module full_adder (A, B, Cin, Sum, Cout);
input A, B, Cin;
output Sum, Cout;
wire S0, C0, C1;
                                        // binding HA modules to module full_adder
    HA H1 (A, B, S0, C0);
    HA H2 (S0, Cin, Sum, C1);
    or (Cout, C0, C1);
endmodule
                                        // Half-Adder Module
module HA (A, B, S, C);
input A, B;
output S, C;
    xor (S, A, B);
    and (C, A, B);
endmodule
```

Review Questions

1. *Write the HDL gate-level description of the priority encoder circuit.*
2. *Write the HDL gate-level hierarchical description of a quadruple 2-to-1 line multiplexer.*
3. *Write the HDL gate-level description of 1-to-4 line demultiplexer.*
4. *Write the HDL data flow and behavioral description of a 4-bit adder/subtractor of unsigned numbers.*
5. *Write the HDL gate-level description of a look-ahead carry generator.*
6. *Write the HDL gate-level and data flow description of the following circuits.*
   i) Binary to BCD converter *ii) BCD to binary converter.*
   iii) BCD to Excess-3 code converter. *iv) Excess-3 to BCD code converter*
   v) Binary to Gray code converter. *vi) Gray to Binary code converter.*
   vii) BCD to Gray code converter
7. *Consider a 4-bit ALU which performs 4 operations. Write an HDL behavioral description for this circuit.*
8. *Write the HDL gate structure description of 3:8 decoder.*
9. *Write the HDL description of the circuit specified by the following Boolean functions. Using continuous assignment statements.*
   a. $Z = AB + ACD + B\overline{C}$
   b. $Y = (B + C)(\overline{B} + \overline{C})(A + \overline{C})$
   c. $Y = \overline{\left[\left(\overline{A} + B\right)\left(C + D\overline{B}\right)\right]}$
10. *Draw the logic diagram of the digital circuit specified by the following HDL description :*

```
module ckt (A, B. C, X, Y);
        input A, B, C;
        output X, Y;
        wire w0;
        and G1 (w0, A, B);
        not G2 (Y, C);
        or G3 (X, w0, Y);
endmodule
```

11. *Write a Verilog model of a full subtractor circuit.* **Dec.-10, Marks 2**

11.6 HDL for Sequential Circuits

11.6.1 Description of D-Latch

Example 11.6.1 *Verilog HDL code in behavioral model of a D-latch.*

Solution : Consider a D-latch with input D and output Q. The D-latch is transparent. It transfers data input D to an output Q as long as the control input is enabled. 'Control' input responds to input signal level so any edge qualifier is not required.

```
module D_latch (D,control,Q);
input D;
input control;
output Q;
reg Q;
always @ (control)
      if (control) Q = D; // If (control=1)
endmodule
```

11.6.2 Description of Flip-Flops

Example 11.6.2 *Verilog HDL code in behavioral model of a D-flip-flop.*

Solution : Consider a positive edge clock D-flip-flop. The D is an input to the flip-flop and Q is an output. Output Q must be declared as 'reg' data type since it is a target output in a procedural assignment statement. When there is positive-edge transition of CLK, input D transfers into Q. At any other time, a change in D will not change Q.

```
module D_ff (D,CLK,Q);
input D,CLK;
output Q;
reg Q;
always @ (posedge CLK)
      Q = D;
endmodule
```

We have seen a description of D-flip-flop with CLK input. But it is necessary to have a reset or preset input for a flip-flop. The reasons is that in the sequential circuits it is necessary to determine the initial state. Without assigning an initial state with an input signal, a sequential circuit can not be tested with HDL simulation. Let us see an example of D-flip-flop with reset input.

Example 11.6.3 *Verilog HDL code in behavioral model of a D-flip-flop with reset input.*

Solution : In this case, the D-flip-flop responds not only to a CLK but also to an asynchronous reset input. The 'CLK' is positive edge trigger and 'Reset' is negative edge trigger. When there is negative edge transition of Reset, as long as Reset is 0, output of flip-flop Q is cleared to 0. If CLK has a positive edge transition and Reset = 1, there is transfer of D into Q synchronously.

```
module D_FF ( D,CLK,Reset,Q);
input D;
```

```
input CLK,Reset;
output Q;
reg Q;
always @ (posedge CLK or negedge Reset)
        if (~Reset) Q = 1'b0;            // If (Reset = 0)
                else Q = D;
endmodule
```

Example 11.6.4 *Verilog HDL code for behavioral model of a JK flip-flop.*

Solution : Here, using values of J,K inputs, the case value ({J, K}) is evaluated and compared with the list of 4 values sequentially. When the value matches, the statement written in front of it, is executed. The output of the JK flip-flop after the application of positive edge clock signal is specified by these four possible conditions.

```
module JK_FF(J,K,CLK,Q,Q̄);
input J,K;
input CLK;
output Q,Q̄;
reg Q;
assign Q̄ = ~Q;
always @ (posedge CLK)
        case ({J,K})
        2'b00:Q = Q;
        2'b01:Q = 1'b0;
        2'b10:Q = 1'b1;
        2'b11:Q = ~Q;
        endcase
endmodule
```

Example 11.6.5 *A positive edge triggered flip-flop has two inputs D_1 and D_2 and a control input that chooses between the two. Write an HDL behavioral description of this flip-flop.*

Solution :

```
module FF(D1, D2, CLK, CN, Q, Qbar)
        input D1, D2, CLK, CN
        output Q, Qbar;
        reg Q, Qbar;
        always @ (Posedge CLK)
        begin
                if (CN==1)
                begin
                        Q = D1
                        Qbar = ~Q;
```

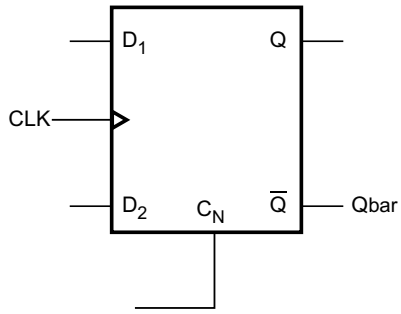


Fig. 11.6.1

```
        end
        else
        begin
            Q = D2
            Qbar = ~Q;
end
endmodule
```

11.6.3 Description of Sequential Circuits

In this section we will discuss the description of two types of sequential circuits, Mealy circuit and Moore circuit.

Example 11.6.6 *Verilog HDL code for given mealy state diagram using behavioral model.*

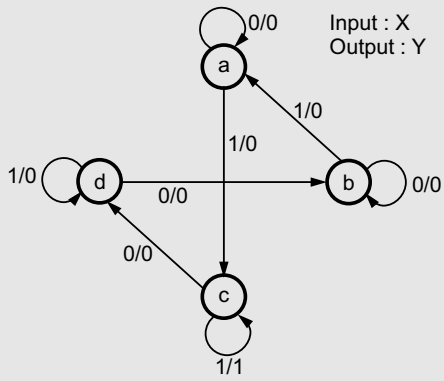


Fig. 11.6.2 State diagram

Solution : Here, X, CLK and Reset are declared as inputs and Y is declared as output. The state of the flip-flops is declared with 2-bit 'reg' type identifiers Prestate (present state) and Nextstate. These 2-bit variables hold the state value of the circuit. The constants can be defined using a keyword 'parameter'. As shown in the state diagram, there are four states a, b, c and d. We have assigned binary 00, 01, 10 and 11 to these states a, b, c and d respectively.

Three always blocks are required in this description. The first always block resets the circuits to the initial state a = 00 if there is negative edge transition of Reset input.

If there is a positive-edge transition of the clock (CLK) and RST= 1, Prestate = Nextstate statement is executed. i.e., on positive-edge clock, any change that occurs in the Nextstate value in the second always block is transferred to Prestate. The second always block obtains Nextstate according to the Prestate (present state) and value of input X. The third always block evaluates the output as a function of present state and input.

```
module Mealy_ckt (X,CLK,Reset,Y);
input X;
```

```
input CLK,Reset;
output Y;
reg Y;
reg [1:0]Prestate,Nextstate;
parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;
always @ (posedge CLK or negedge Reset)
        if (~Reset)Prestate = a; // Reset the ckt to a.
        else Prestate = Nextstate;
always @ (Prestate or X) // Obtain next state
        case (Prestate)
                a : if(X)       Nextstate   = c;
                    else        Nextstate   = a;
                b : if(X)       Nextstate   = a;
                    else        Nextstate   = b;
                c : if(X)       Nextstate   = c;
                    else        Nextstate   = d;
                d : if(X)       Nextstate   = d;
                  else          Nextstate   = b;
        endcase
always @ (Prestate or X) // Evaluate output
          case (Prestate)
                a : Y = 0;
                b : Y = 0;
                c : if(X) Y = 1'b1;else Y = 1'b0;
                d : Y = 0;
          endcase
endmodule
```

Example 11.6.7 *Verilog HDL code for given Moore state diagram using behavioral model.*

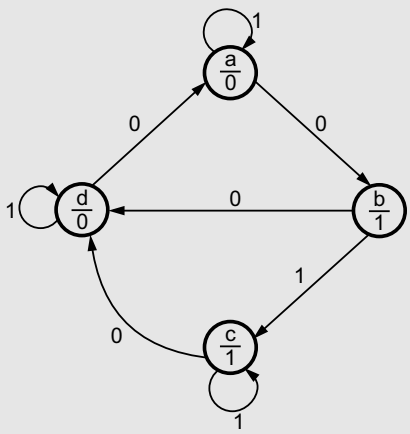


Fig. 11.6.3

Solution : Here X, CLK and Reset are declared as inputs and Y is declared as output. The present state of the circuit is declared as Prestate. The state transitions can be specified within an always block. The always block resets the circuit to the initial state a = 00 if there is a negative edge transition of Reset input. If there is a positive edge transition of the clock (CLK) and RST = 1, the state transitions occur.

In case of Moore circuit, output of the circuit is independent of the input. It is taken directly from the outputs of the flip-flops. In the second always block according to the present state, output Y can be determined.

```
module Moore_ckt (X,CLK,Reset,Y);
input X;
input CLK,Reset;
output Y;
reg [1:0]Prestate;
parameter a = 2'b00, b = 2'b01, c = 2'b10, d = 2'b11;
always @ (posedge CLK or negedge Reset)
            if (~ Reset) Prestate = a; // Reset to state a
            else
            case (Prestate)
              a : if(X)Prestate = a; else prestate = b;
              b : if(X)Prestate = c; else prestate = d;
              c : if(X)Prestate = c; else prestate = d;
              d : if(X)Prestate = d; else prestate = a;
            endcase
always @ (Prestate) // Determine output
            case (Prestate)
              a : Y = 0;
              b : Y = 1;
              c : Y = 1;
              d : Y = 0;
            endcase
endmodule
```

Example 11.6.8 *Write HDL code for the following Mealy state diagram.*

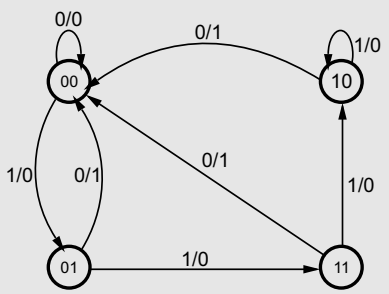


Fig. 11.6.4

Solution :

```
module Mealy_ckt (X, CLK, Reset, Y);
input X;
input CLK, Reset;
output Y;
reg Y;
reg [1:0] Prestate, Nextstate;
Parameter a = 2'b00, b = 2'b01, c = 2'b10, c = 2'b10, d = 2'b11;
always @ (posedge CLK or negedge Reset)
if (~Reset) Prestate = a;
else Prestate = Nextstate;
always @ (Prestate or X)
        case (Prestate)
                a : if(X)       Nextstate = b
                        else            Nextstate = a
                b : if(X)       Nextstate = d
                        else            Nextstate = a
                c : if(X)       Nextstate = c
                        else            Nextstate = a
                d : if(X)       Nextstate = c
                        else            Nextstate = a
        endcase
always @ (Prestate or X)
        case (Prestate)
                a:Y = 0;
                b:if(X) y = 1'b0 ;  else Y = 1'b1 ;
                        c:if(X)  y = 1'b0 ;  else Y = 1'b1 ;
                        d:if(X)  y = 1'b0 ;  else Y = 1'b1 ;
                end case
                end module
```

11.6.4 HDL for Registers and Counters

Gate-level (or structural) and behavioral modeling techniques can be used for describing registers and counters in Verilog HDL. This section explains these descriptions.

In gate-level (or structural level), the circuit is considered as a collection of components such as gates, flip-flops and multiplexers. According to the logic diagram representation of the circuit, a hierarchical model of the design is formed. The required components are instantiated in this hierarchical model or description of the circuit.

In the behavioral model, the various operations which the circuit performs are described. The function table of the circuit is used for describing the circuit using this technique.

Example 11.6.9 *Verilog HDL code in structural model of a buffer register.*

Solution : Refer Fig. 11.6.5 The figure shows that the buffer register is constructed with four D-flip-flops. There are two modules in this example. The first module declares the inputs, outputs and then instantiates the stages of the register. There are four D-flip-flops as shown in the logic diagram. So four instantiations are required which specify the interconnections between the four stages. The second module has one always block which describes the D-flip-flop. This module for flip-flop defines one stage of the register.

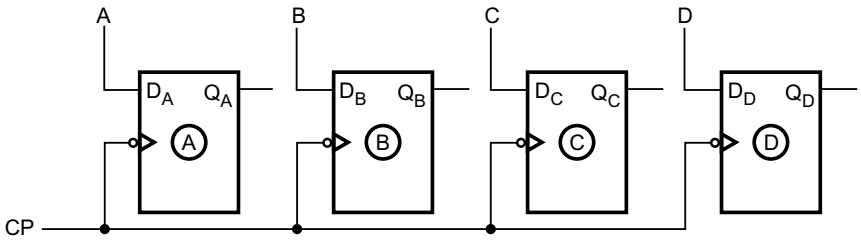


Fig. 11.6.5 Buffer register

```
module buf_reg (A,B,C,D,CP,QA,QB,QC,QD);
input A,B,C,D;              // Parallel input
input CP;                   // Clock input
output QA,QB,QC,QD;         // Parallel output
      // Instantiate the four stages
       STG STG0  (A,CP,QA);
       STG STG1  (B,CP,QB);
       STG STG2  (C,CP,QC);
       STG STG3  (D,CP,QD);
endmodule
      // One stage of D-flip-flop
module STG (D,CP,Q);
input D;
input CP;
output Q;
reg Q;
reg D;
      always @ (negedge CP)
            Q = D
endmodule
```

Example 11.6.10 *Verilog HDL code in structural model of a universal shift register.*

Solution : The circuit is shown in the Fig. 11.6.6. As shown in this logic diagram, a universal shift register is constructed with four multiplexers and four D-flip-flops. The two modules are required in the structural model of a universal shift register. The first module declares the inputs, outputs and then instantiates stages of the register. The four instantiations are required which specify the interconnections between the four stages. According to the logic diagram, the detail construction of the register is provided by these stages. The second module has two **always** blocks. The first **always** block describes the 4-to-1 multiplexer and the second block describes the D-flip-flop. One stage of the register consists of one 4-to-1 multiplexer and one D-flip-flop.

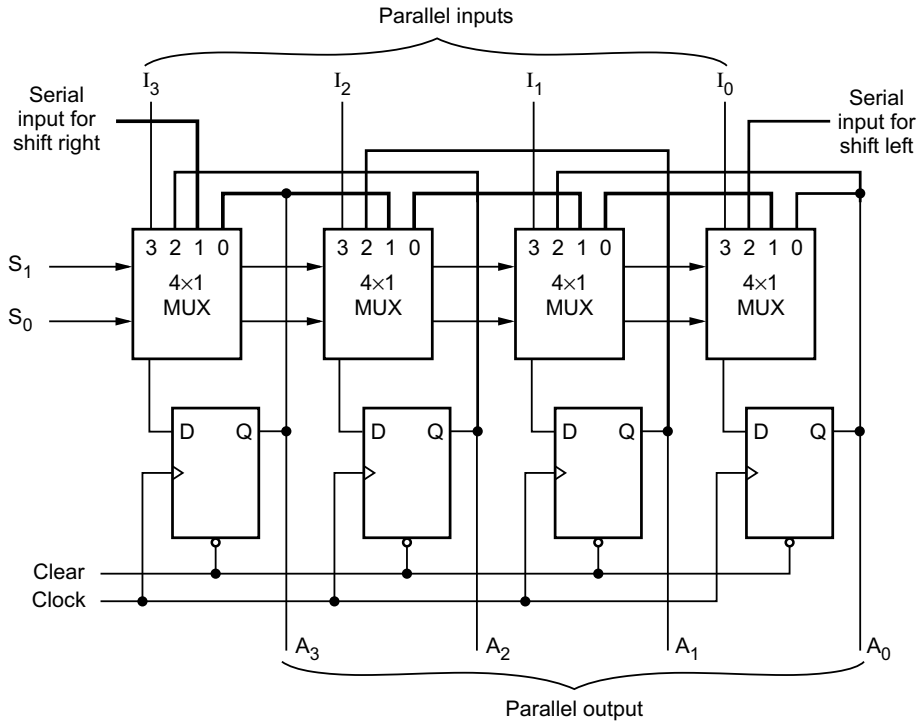


Fig. 11.6.6 4-bit universal shift register

Mode control		Register operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Table 11.6.1 Mode control and register operation

```
module Unishft_reg (I,select,LFin,RTin,CLK,CLR,A);
input [3:0]I;       // Parallel input
input [1:0]select   // Select mode
input LFin,RTin,CLK,CLR;  // Serial inputs,clock,clear
output [3:0]A;      // Parallel output
      // Instantiate the four stages
      // Each stage consists of 4-to-1 multiplexer and D_FF
        STG STG0(A[0],A[1],LFin,I[0],select,CLK,CLR,A[0]);
        STG STG1(A[1],A[2],A[0],I[1],select,CLK,CLR,A[1]);
        STG STG2(A[2],A[3],A[1],I[2],select,CLK,CLR,A[2]);
        STG STG3(A[3],RTin,A[2],I[3],select,CLK,CLR,A[3]);
endmodule
      // One stage of shift register (4-to-1 multiplexer and D_FF)
module STG(i0,i1,i2,i3,select,CLK,CLR,Q);
input i0,i1,i2,i3,CLK,CLR;
input [1:0]select;
output Q;
reg Q;
reg D;
      // 4-to-1 multiplexer
always @ (i0 or i1 or i2 or i3 or select)
      case (select)
            2'b00:D = i0;
            2'b01:D = i1;
            2'b10:D = i2;
            2'b11:D = i3;
      endcase
      // D flip-flop
always @ (posedge CLK or negedge CLR)
            if (~CLR)Q = 1'b0;
            else Q = D;
endmodule
```

Example 11.6.11 *Verilog HDL code in behavioral model of a universal shift register.*

Solution : As shown in Fig. 11.6.7. there are two selection inputs, S0, S1, two serial inputs, for shift left, for shift right, a 4-bit parallel input (I_3 - I_0) and a 4-bit parallel output (A_3 - A_0). The CLR is active low input. A low of CLR signal clears the register asynchronously. The CLR must be high for the operations which get performed when positive edge CLK is present. These four operations are given in Table 11.6.1. It is to be noted that the S1 and S0 are concatenated into a 2-bit vector after the keyword, 'case'. The four cases are,

1. When S1 S0 = 0 0, the contents of a register remain unchanged i.e. A = A.
2. When S1 S0 = 0 1, shift right operation takes place. i.e. A = {RTin, A [3 : 1]} Here, the contents of a register are given by the concatenation of the serial input, RTin and three flip-flops (A_3-A_1).
3. When S1 S0 = 1 0, shift left operation is performed. i.e. A = {A [2 : 0], LF in}.
4. When S1 S0 = 1 1, register loaded with parallel input. i.e. A = PIin.

The always block describes all the above functions which can be performed by the register. The function of the circuit is described without considering any particular hardware.

```
                        // Behavioral model of universal shift register
                        // The circuit is shown in the Fig. 11.6.6.
                        // Refer Table 11.6.1.
module                  Unishft_reg(S1,S0,PIin,LFin,RTin,CLK,CLR,A);
input S1,S0;            // Select inputs
input LFin,RTin;        // Serial inputs
input CLK,CLR;          // Clock and clear
input [3:0]PIin;        // Parallel input I3-I0
output      [3:0]A;     // Register output A3-A0
always @ (posedge CLK or negedge CLR)
      if (~CLR)A = 4'b0000;
      else
      case ({S1,S0})
      2'b00 : A = A;                  // No change
      2'b01 : A = {RTin,A[3:1]};      // Shift right
      2'b10 : A = {A[2:0],LFin};      // Shift left
      2'b11 : A = PIin,           // Parallel load input
      endcase
endmodule
```

Example 11.6.12 *Verilog HDL code for 4-bit synchronous counter with parallel load.*

Solution : The circuit of a 4-bit binary counter with parallel load is shown in the Fig. 11.6.7. There are four inputs, count, load, clock and clear which determine the operation of the counter. Table 11.6.2 is the function table for this circuit. The counter has four data inputs (I_3-I_0), four data outputs (A_3-A_0) and a carry output (CO). The CO is generated with an AND gate. So we can use '&' operator and an assign statement for this combinational circuit. If Count = 1, Load = 1 and A = 1111, then CO = 1; otherwise CO = 0. When positive edge clock or negative edge CLR appears, the operations which get performed are included in the **always** block. The CLR input is active low. So a negative signal on CLR clears A. i.e. A = 0000. If CLR = 1 and positive edge is present on CLK, then one of the following operations takes place.

1. If Load = 1, input (I_3-I_0) is loaded into A.
2. If Load = 0 and Count = 1, A is incremented by 1.
3. If Load = 1 and Count = 0, the A remains unchanged.

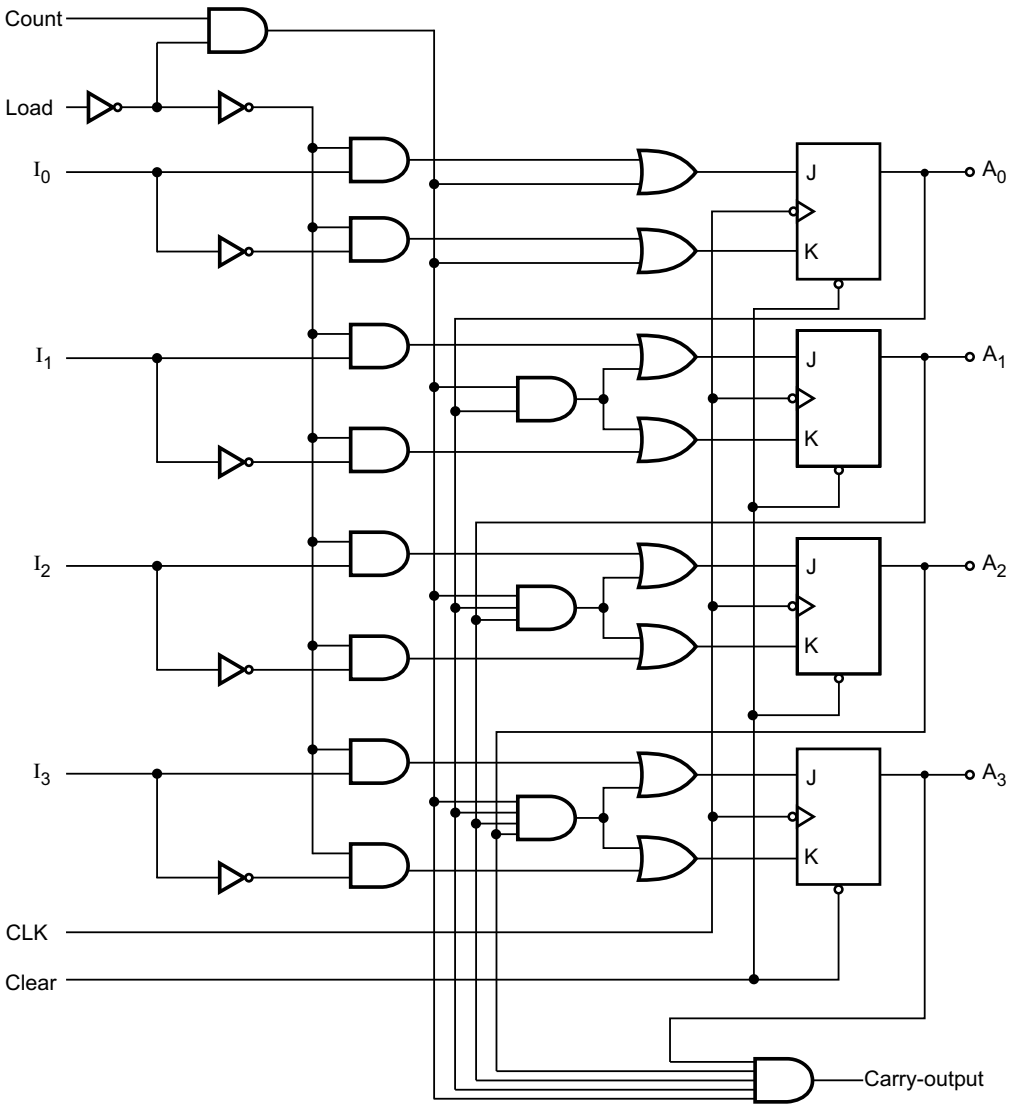


Fig. 11.6.7

Clear	CLK	Load	Count	Function
0	X	X	X	Clear to 0
1	↑	1	X	Load inputs
1	↑	0	1	Count next binary state
1	↑	0	0	No change

Table 11.6.2

```
module bin_ctr (Count,Load,I,CLK,CLR,A,CO);
input Count, Load;
input CLK,CLR;
input [3:0]I;          // Data input
output[3:0]A;              // Data output
output CO;          // Output carry
reg [3:0]A;
assign CO = Count & ~ Load & (A = = 4'b1111);
always @ (posedge CLK or negedge CLR)
      if (~ CLR)A = 4'b0000;
      else if (Load)A = I;
      else if (Count)A = A+1'b1;
      else  A = A;
endmodule
```

Example 11.6.13 *Verilog HDL code in structural model of a ripple counter.*

Solution : The circuit is shown in the Fig. 11.6.8. The figure shows that the ripple counter is constructed with four D-flip-flops. There are two modules in this example. The first module declares the inputs, outputs and then instantiates the stages of the counter. There are four D-flip-flops as shown in the logic diagram. So four instantiations are required which specify the interconnections between the four stages and provide the detail construction of a counter. The second module has one **always** block which describes complementing flip-flop with delay. The circuit of a complementing flip-flop is obtained by connecting the complement output to D-input. This module for complementing flip-flop defines one stage of the counter. A delay of 2 time units is given to the flip-flop which defines the time from the instant that the clock is applied to the instant that the flip-flop complements.

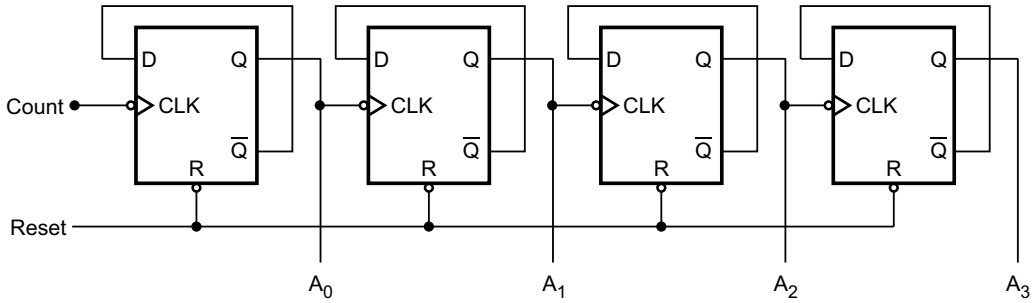


Fig. 11.6.8 4-bit binary ripple counter using D flip-flop

```
module ripple_ctr (Count,Reset,A0,A1,A2,A3);
input Count,Reset;
output A0,A1,A2,A3;
      // Instantiate complementing flip-flop
       STG  STG0 (Count,Reset,A0);
       STG  STG1 (A0,Reset,A1);
       STG  STG2 (A1,Reset,A2);
       STG  STG3 (A2,Reset,A3);
endmodule
      // One stage of a counter
      // Complementing flip-flop with delay
module STG(CLK,Reset,Q);
input CLK,Reset;
output Q;
reg Q;
always @ (negedge CLK or negedge Reset)
      if (~Reset)Q = 1'b0;
      else Q = #2(~Q);  // Delay of 2 time units
endmodule
```

Example 11.6.14 *Design 2-bit up/down counter using verilog.* **Dec.-12, Marks 8**

Solution :

```
      module      updowncount (clock, E, up_down, Q);
      input       clock, E, up_down;
      output reg  [1:0] Q;
      integer     direction;
      always      @ (posedge clock)
      begin
            if    (up_down)        // check direction of counter
            direction = 1;
```

```
else
        direction = – 1;
if (E)                          // check if counter is enabled
        Q <= Q + direction;
end
endmodule
```

Review Questions

1. *Give the HDL behavioral description for the following circuits.*
   i) D Latch ii) D flip-flop iii) JK flip-flop
2. *Give the HDL description for the circuits having state diagram as shown in the Fig. 11.6.9. given below.*

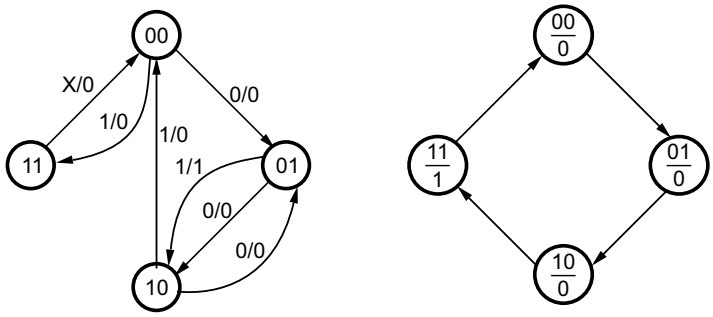


Fig. 11.6.9

3. *Write the HDL structural description of a controlled buffer register.*
4. *Draw and write the HDL structural and behavioral descriptions of the following circuits.*
   a) Parallel in parallel out shift register. b) 10-bit ring counter.
   c) Johnson counter. d) 4-bit pseudo-random sequence generator.
   e) Two-bit asynchronous binary counter. f) Three-bit synchronous binary counter.
5. *Give behavioral and structural descriptions of universal shift register.*
6. *Give behavioral and structural descriptions of 4-bit synchronous binary counter with parallel load.*

❑❑❑

APPENDIX A

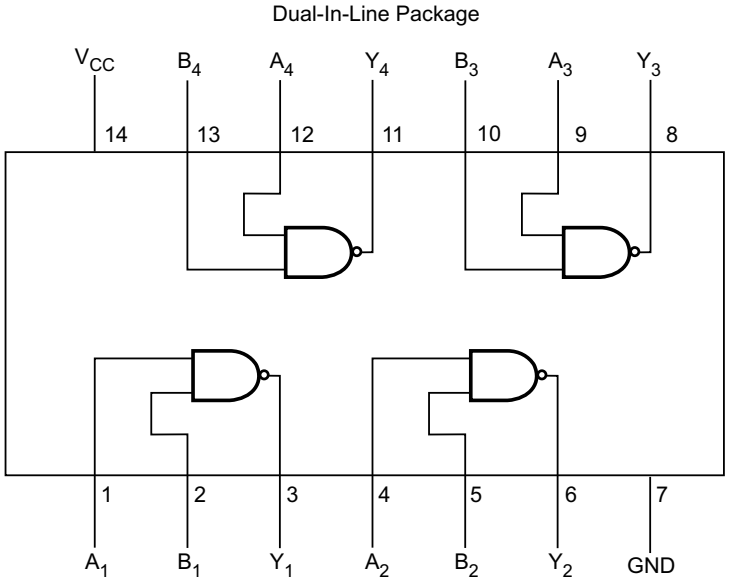
Data Sheets For Commonly Used Digital ICs

DM54LS00/DM74LS00 Quad 2-Input NAND Gates

GENERAL DESCRIPTION

This device contains four independent gates each of which perform the logic NAND function.

CONNECTION DIAGRAM



FUNCTION TABLE

$$Y = \overline{AB}$$

Inputs		Output
A	**B**	**Y**
L	L	H
L	H	H
H	L	H
H	H	L

H = High Logic Level

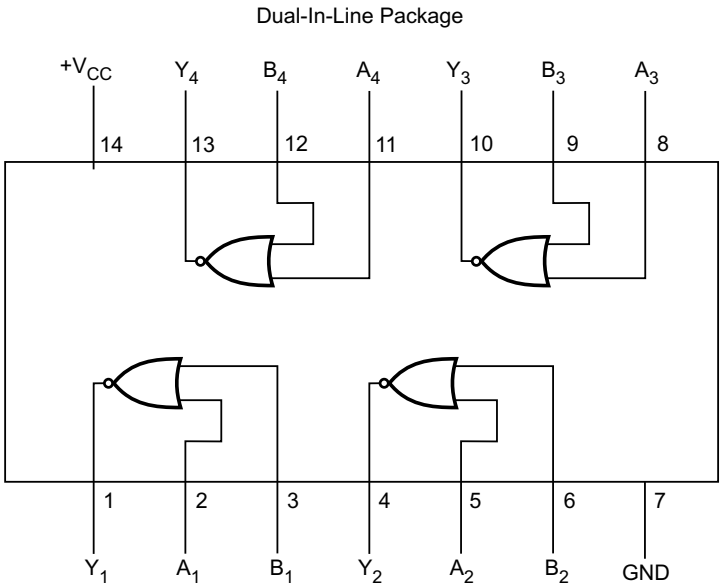
L = Low Logic Level

DM54LS02/DM74LS02 Quad 2-Input NOR Gates

GENERAL DESCRIPTION

This device contains four independent gates each of which performs the logic NOR function.

CONNECTION DIAGRAM



FUNCTION TABLE

$$Y = \overline{A + B}$$

Inputs		Output
A	**B**	**Y**
L	L	H
L	H	L
H	L	L
H	H	L

H = High Logic Level

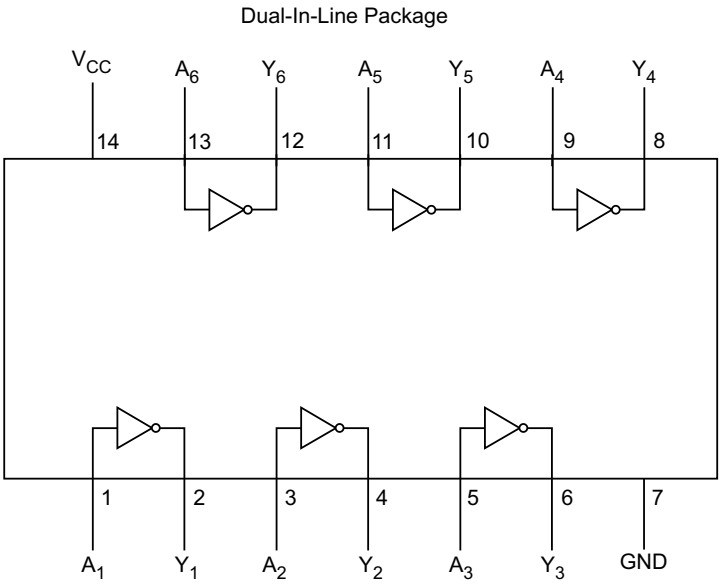
L = Low Logic Level

DM54LS04/DM74LS04 Hex Inverting Gates

GENERAL DESCRIPTION

This device contains six independent gates each of which performs the logic INVERT function.

CONNECTION DIAGRAM



FUNCTION TABLE

$Y = \overline{A}$

Inputs	Output
A	Y
L	H
H	L

H = High Logic Level

L = Low Logic Level

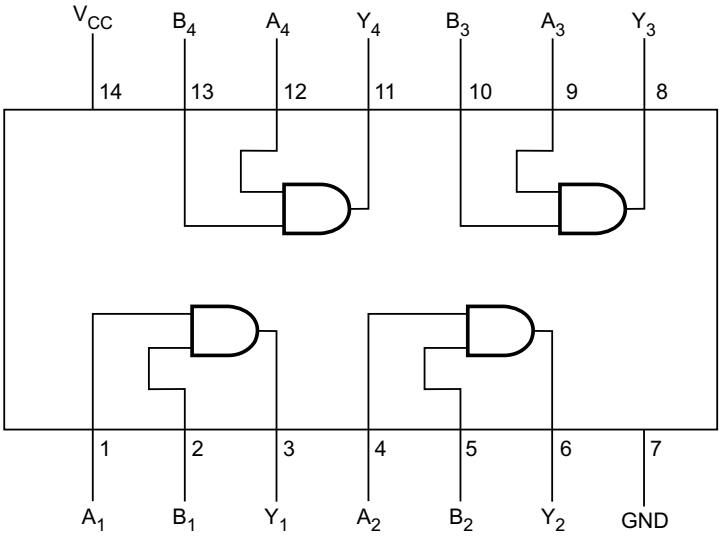
DM54LS08/DM74LS08 Quad 2-Input AND Gates

GENERAL DESCRIPTION

This device contains four independent gates each of which performs the logic AND function.

CONNECTION DIAGRAM

Dual-In-Line Package



FUNCTION TABLE

Y = AB

Inputs		Output
A	**B**	**Y**
L	L	L
L	H	L
H	L	L
H	H	H

H = High Logic Level

L = Low Logic Level

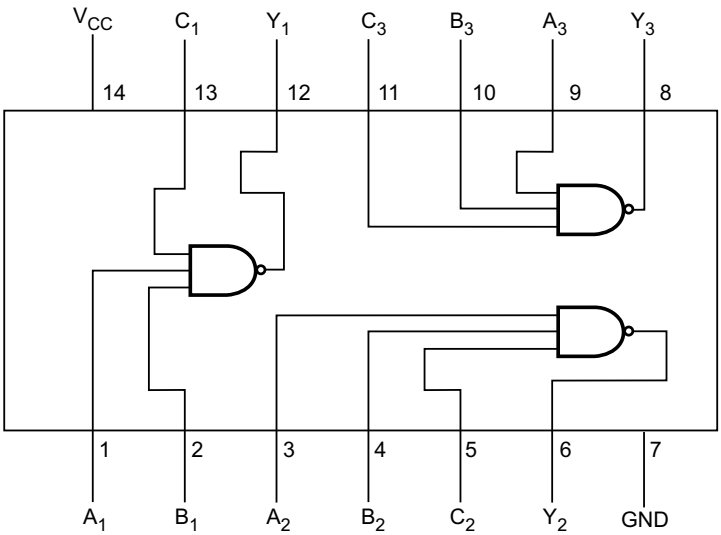
DM54LS10/DM74LS10 Triple 3-Input NAND Gates

GENERAL DESCRIPTION

This device contains three independent gates each of which performs the logic NAND function.

CONNECTION DIAGRAM

Dual-In-Line Package



FUNCTION TABLE

$$Y = \overline{ABC}$$

Inputs			Output
A	B	C	Y
X	X	L	H
X	L	X	H
L	X	X	H
H	H	H	L

H = High Logic Level

L = Low Logic Level

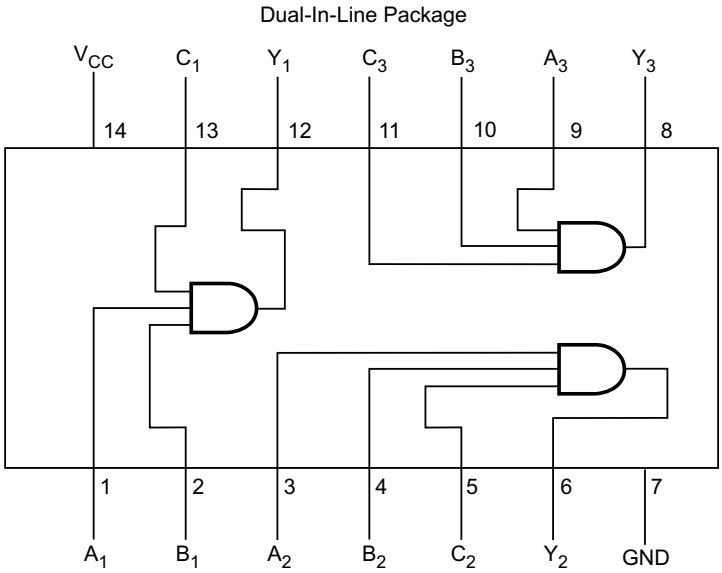
X = Either Low or High Logic Level

DM54LS11/DM74LS11 Triple 3-Input AND Gates

GENERAL DESCRIPTION

This device contains three independent gates each of which performs the logic AND function.

CONNECTION DIAGRAM



FUNCTION TABLE

Y = ABC

Inputs			Output
A	**B**	**C**	**Y**
X	X	L	L
X	L	X	L
L	X	X	L
H	H	H	H

H = High Logic Level

L = Low Logic Level

X = Either Low or High Logic Level

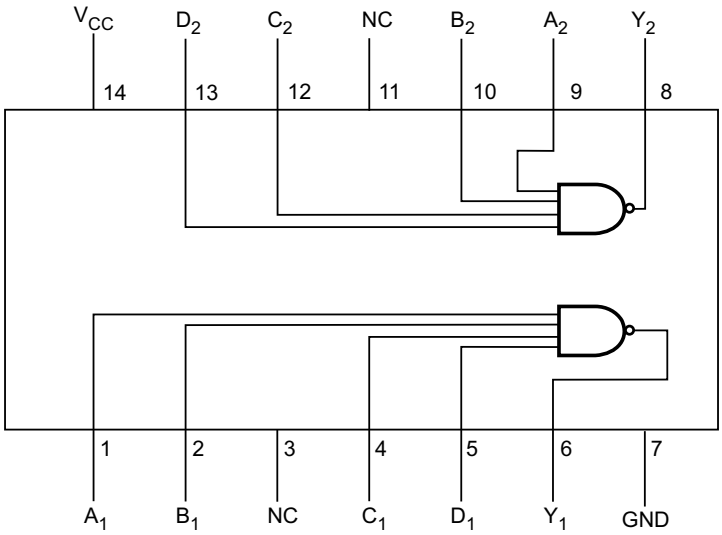
DM54LS20/DM74LS20 Dual 4-Input NAND Gates

GENERAL DESCRIPTION

This device contains two independent gates each of which performs the logic NAND function.

CONNECTION DIAGRAM

Dual-In-Line Package



FUNCTION TABLE

$$Y = \overline{ABCD}$$

Inputs				Output
A	B	C	D	Y
X	X	X	L	H
X	X	L	X	H
X	L	X	X	H
L	X	X	X	H
H	H	H	H	L

H = High Logic Level

L = Low Logic Level

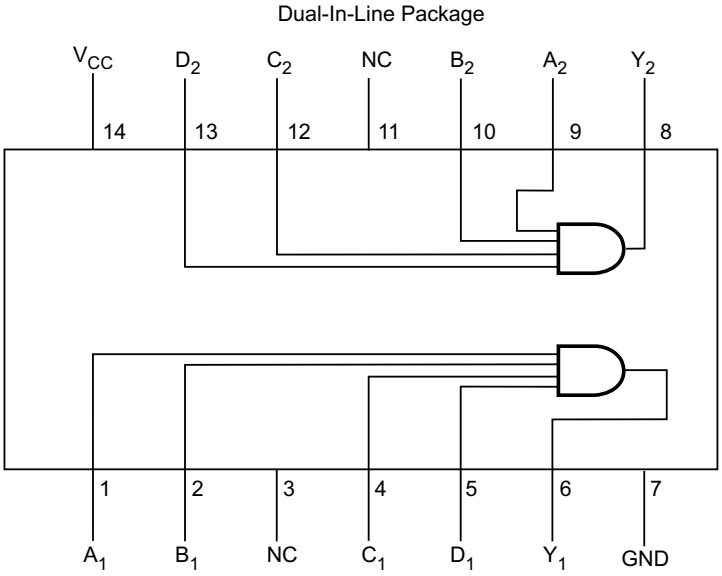
X = Either Low or High Logic Level

DM54LS21/DM74LS21 Dual 4-Input AND Gates

GENERAL DESCRIPTION

This device contains two independent gates each of which performs the logic AND function.

CONNECTION DIAGRAM



FUNCTION TABLE

Y = ABCD

Inputs				Output
A	**B**	**C**	**D**	**Y**
X	X	X	L	L
X	X	L	X	L
X	L	X	X	L
L	X	X	X	L
H	H	H	H	H

H = High Logic Level

L = Low Logic Level

X = Either Low or High Logic Level

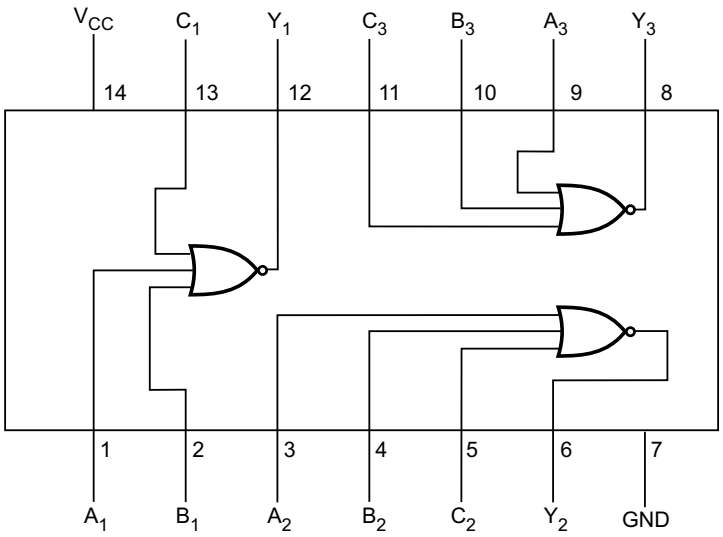
DM54LS27/DM74LS27 Triple 3-Input NOR Gates

GENERAL DESCRIPTION

This device contains three independent gates each of which performs the logic NOR function.

CONNECTION DIAGRAM

Dual-In-Logic Package



FUNCTION TABLE

$$Y = \overline{A + B}$$

Inputs		Output
A	**B**	**Y**
L	L	H
L	H	L
H	L	L
H	H	L

H = High Logic Level

L = Low Logic Level

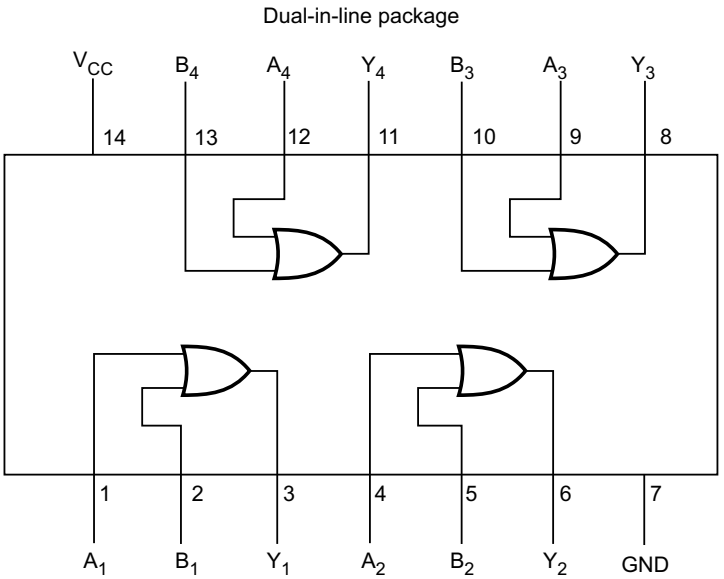
x = Either Low or High Logic Level

DM54LS32/DM74LS32 Quad 2-Input OR Gates

GENERAL DESCRIPTION

This device contains four independent gates each of which performs the logic OR function.

CONNECTION DIAGRAM



FUNCTION TABLE

Y = A + B

Inputs		Output
A	**B**	**Y**
L	L	L
L	H	H
H	L	H
H	H	H

H = High Logic Level

L = Low Logic Level

DECODERS 54/7442, LS42

BCD-To-Decimal Decoder (1-of-10)

- **Mutually exclusive outputs**
- **1-of-8 demultiplexing capability**
- **Outputs disabled for input codes above nine**

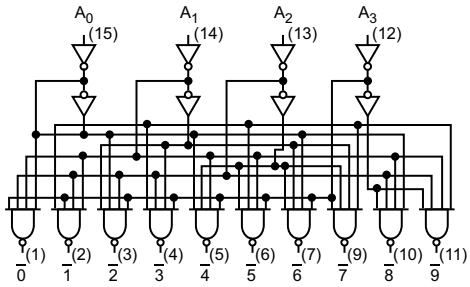
DESCRIPTION

The '42 decoder accepts four active HIGH BCD inputs and provided 10 mutually exclusive active LOW outputs, as shown by logic symbol or diagram. The active LOW outputs facilitate addressing other MSI units with active LOW input enables.

The logic design of the '42 ensures that all outputs are HIGH when binary codes greater than nine are applied to the inputs.

The most significant input, A_3, produces a useful inhibit function when the '42 is used as a 1-of-8 decoder. The A_3 input can also be used as the Data Input in an 8-output demultiplexer application.

LOGIC DIAGRAM

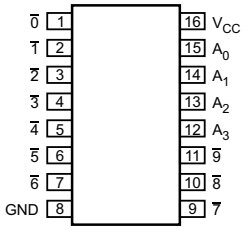


( ) = Pin number
V_{CC} = Pin 16
GND = Pin 8

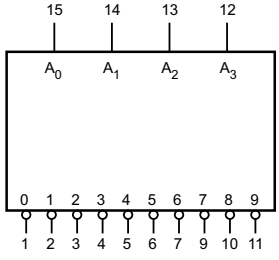
FUNCTION TABLE

A_3	A_2	A_1	A_0	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$	$\overline{8}$	$\overline{9}$
L	L	L	L	L	H	H	H	H	H	H	H	H	H
L	L	L	H	H	L	H	H	H	H	H	H	H	H
L	L	H	L	H	H	L	H	H	H	H	H	H	H
L	L	H	H	H	H	H	L	H	H	H	H	H	H
L	H	L	L	H	H	H	H	L	H	H	H	H	H
L	H	L	H	H	H	H	H	H	L	H	H	H	H
L	H	H	L	H	H	H	H	H	H	L	H	H	H
L	H	H	H	H	H	H	H	H	H	H	L	H	H
H	L	L	L	H	H	H	H	H	H	H	H	L	H
H	L	L	H	H	H	H	H	H	H	H	H	H	L
H	L	H	L	H	H	H	H	H	H	H	H	H	H
H	L	H	H	H	H	H	H	H	H	H	H	H	H
H	H	L	L	H	H	H	H	H	H	H	H	H	H
H	H	L	H	H	H	H	H	H	H	H	H	H	H
H	H	H	L	H	H	H	H	H	H	H	H	H	H
H	H	H	H	H	H	H	H	H	H	H	H	H	H

PIN CONFIGURATION

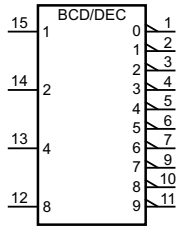


LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

LOGIC SYMBOL (IEEE/IEC)



DM5446A/DM7446A, DM5447A/DM7447A BCD to 7-segment Decoders/Drivers

GENERAL DESCRIPTION

The 46A and 47A feature active-low outputs designed for driving common-anode LEDs or incandescent indicators directly. All of the circuits have full ripple-blanking input/output controls and a lamp test input. Segment identification and resultant displays are shown on a following page. Display patterns for BCD input count above nine are unique symbols to authenticate input conditions.

All of the circuits incorporate automatic leading and/or trailing-edge, zero-blanking controls (RBI and RBO). Lamp test (LT) of these devices may be performed at any time when the BI/RBO node is at a high logic level. All types contain an overriding blanking input (BI) which can be used to control the lamp intensity (by pulsing) or to inhibit the outputs.

FEATURES

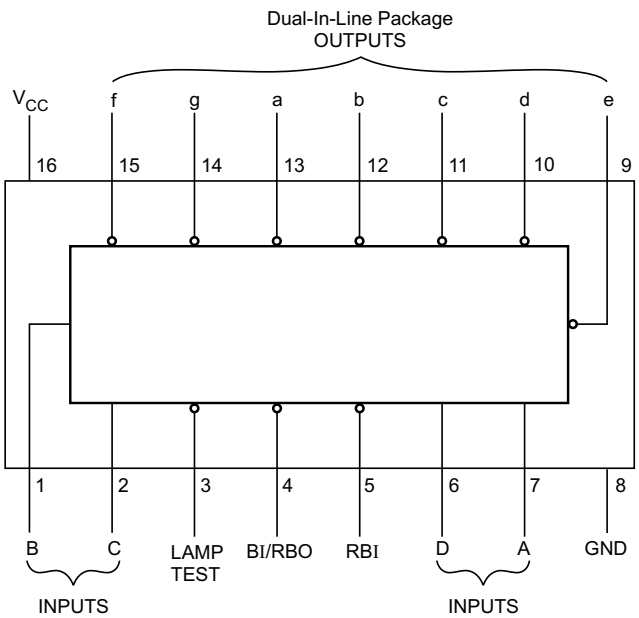
All circuit types feature lamp intensity modulation capability

Open-collector outputs drive indicators directly

Lamp-test provision

Leading/trailing zero suppression

CONNECTION DIAGRAM



FUNCTION TABLE

46A,74A

Decimal or Function	Inputs						BI/RBO (Note 1)	Outputs							Note
	LT	RBI	D	C	B	A		a	b	c	d	e	f	g	
0	H	H	L	L	L	L	H	L	L	L	L	L	L	H	
1	H	X	L	L	L	H	H	H	L	L	H	H	H	H	
2	H	X	L	L	H	L	H	L	L	H	L	L	H	L	
3	H	X	L	L	H	H	H	L	L	L	L	H	H	L	
4	H	X	L	H	L	L	H	H	L	L	H	H	L	L	
5	H	X	L	H	L	H	H	L	H	L	L	H	L	L	
6	H	X	L	H	H	L	H	H	H	L	L	L	L	L	
7	H	X	L	H	H	H	H	L	L	L	H	H	H	H	(2)
8	H	X	H	L	L	L	H	L	L	L	L	L	L	L	
9	H	X	H	L	L	H	H	L	L	L	H	H	L	L	
10	H	X	H	L	H	L	H	H	H	H	L	L	H	L	
11	H	X	H	L	H	H	H	H	H	L	L	H	H	L	
12	H	X	H	H	L	L	H	H	L	H	H	H	L	L	
13	H	X	H	H	L	H	H	L	H	H	L	H	L	L	
14	H	X	H	H	H	L	H	H	H	H	L	L	L	L	
15	H	X	H	H	H	H	H	H	H	H	H	H	H	H	
BI	X	X	X	X	X	X	L	H	H	H	H	H	H	H	(3)
RBI	H	L	L	L	L	L	L	H	H	H	H	H	H	H	(4)
LT	L	X	X	X	X	X	H	L	L	L	L	L	L	L	(5)

Note 1 : BI/RBO is a wire-AND logic serving as blanking input (BI) and / or ripple-blanking output (RBO).

Note 2 : The blanking input (BI) must be open or held at a high logic level when output functions 0 through 15 are desired. The ripple-blanking input (RBI) must be open or high if blanking of a decimal zero is not desired.

Note 3 : When a low logic level is applied directly to the blanking input (BI), all segment outputs are high regardless of the level of any other input.

Note 4 : When ripple-blanking input (RBI) and inputs A, B, C and D are at a low level with the lamp test input high, all segment outputs go H and the ripple-blanking output (RBO) goes to a low level (response condition).

Note 5 : When the blanking input/ripple-blanking output (BI/RBO) is open or held high and a low is applied to the lamp-test input, all segment outputs are L.

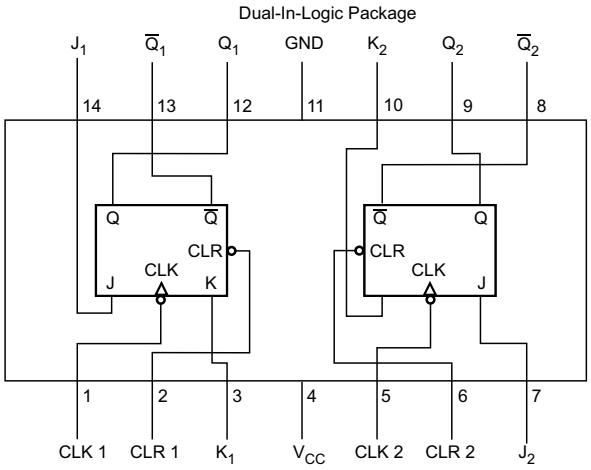
H = High level, L = Low level, X = Don't Care

DM54LS73A/DM74LS73A Dual Negative-Edge-Triggered Master-Slave J-K Flip-Flops with Clear and Complementary Outputs

GENERAL DESCRIPTION

This device contains two independent negative-edge-triggered J-K flip-flops with complementary outputs. The J and K data is processed by the flip-flops on the falling edge of the clock pulse. The clock triggering occurs at a voltage level and is not directly related to the transition time of the negative going edge of the clock pulse. The data on the J and K inputs is allowed to change while the clock is high or low without affecting the outputs as long as setup and hold times are not violated. A low logic level on the clear input will reset the outputs regardless of the levels of the other inputs.

CONNECTION DIAGRAM



FUNCTION TABLE

Inputs				Output	
CLR	**CLK**	**J**	**K**	**Q**	**$\overline{Q}$**
L	X	X	X	L	H
H	↓	L	L	Q_0	$\overline{Q}_0$
H	↓	H	L	H	L
H	↓	L	H	L	H
H	↓	H	H	Toggle	
H	H	X	X	Q_0	$\overline{Q}_0$

H = High Logic Level

L = Low Logic Level

X = Either Low or High Logic Level

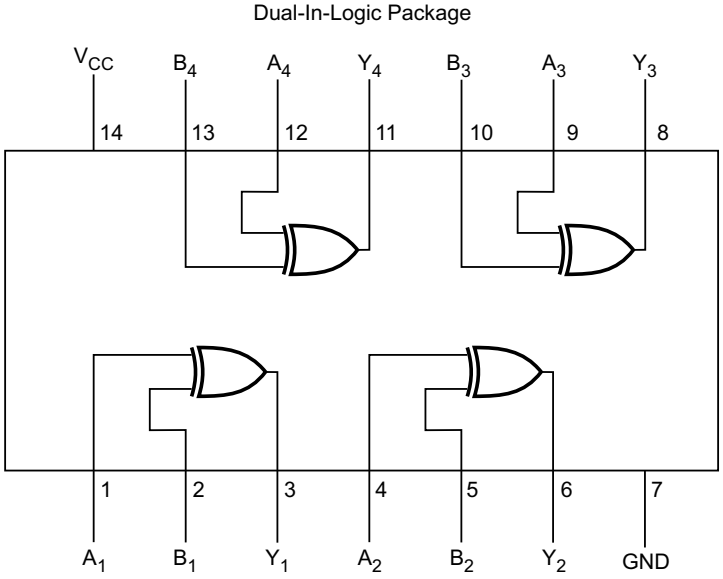
↓ = Negative going-edge of pulse

DM54LS86/DM74LS86 Quad 2-Input Exclusive-OR Gates

GENERAL DESCRIPTION

This device contains four independent gates each of which performs the logic exclusive-OR function.

CONNECTION DIAGRAM



FUNCTION TABLE

$$Y = A \oplus B = \overline{A}B + A\overline{B}$$

Inputs		Output
A	**B**	**Y**
L	L	L
L	H	H
H	L	H
H	H	L

H = High Logic Level

L = Low Logic Level

FLIP-FLOPS 54/7474, LS74A, S74

DESCRIPTION Dual D-Type Flip-Flop

The '74 is a dual positive edge-triggered D-type flip-flop featuring individual Data, Clock, Set and Reset input; also complementary Q and $\overline{Q}$ outputs.

TYPE	TYPICAL f_{MAX}	TYPICAL SUPPLY CURRENT (Total)
7474	25MHz	17mA
74LS74A	33MHz	4mA

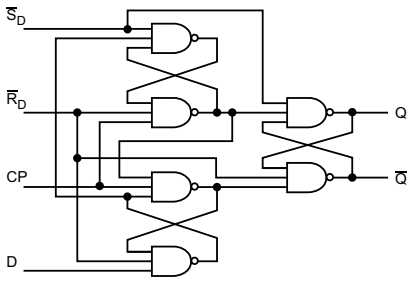
Set $(\overline{S}_D)$ and Reset $(\overline{R}_D)$ are asynchronous active-LOW inputs and operate independently of the Clock Input. Information on the Data (D) input is transferred to the Q output on the LOW-to-HIGH transition of the clock pulse. The D inputs must be stable one setup time prior to the LOW-to-HIGH clock transition for predictable operation. Although the Clock Input is level-sensitive, the positive transition of the clock pulse between the 0.8 V and 2.0 V levels should be equal to or less than the clock-to-output delay time for reliable operation.

MODE SELECT-FUNCTION TABLE

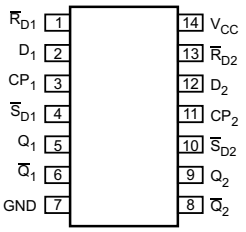
OPERATING MODE	INPUTS				OUTPUTS	
	$\overline{S}_D$	$\overline{R}_D$	CP	D	Q	$\overline{Q}$
Asynchronous Set	L	H	X	X	H	L
Asynchronous Reset (Clear)	H	L	X	X	L	H
Undetermined [a]	L	L	X	X	H	H
Load "1" (Set)	H	H	↑	h	H	L
Load "0" (Reset)	H	H	↑	↑	L	H

H = HIGH voltage level steady state.
h = HIGH voltage level one setup time prior to the LOW-to-HIGH clock transition.
L = LOW voltage level steady state.
↑ = LOW voltage level one setup time prior to the LOW-to-HIGH clock transition.
X = Don't care.
↑ = LOW-to-HIGH clock transition.
NOTE

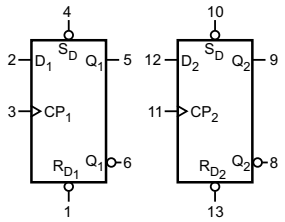
LOGIC DIAGRAM



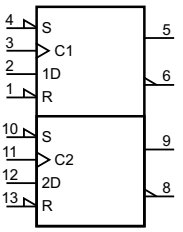
PIN CONFIGURATION



LOGIC SYMBOL



LOGIC SYMBOL (IEEE/IEC)



LATCHES 54/7475, LS75

Quad Bistable Latch

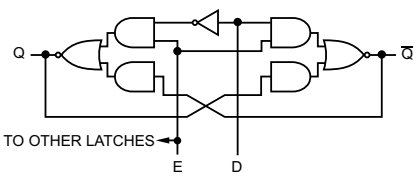
- **4-bit bistable latch**
- **Refer to 54LS/74LS375 for V_{CC} and GND on corner pins**

TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (Total)
7475	18ns (t_{PLH}) 9ns (t_{PHL})	32mA

DESCRIPTION

The '75 has four biastable latches. Each 2-bit latch is controlled by an active HIGH Enable input (E). When E is HIGH, the data enters the latch and appears at the Q output. The Q outputs follow the Data inputs as long as E is HIGH.The data on the D inputs one setup time before the HIGH-to-LOW transition of the enable will be stored in the latch. The latched outputs remain stable as long as the enable is LOW.

LOGIC DIAGRAM (Each Latch)



MODE SELECT-FUNCTION TABLE

OPERATING MODE	INPUTS		OUTPUTS	
	E	D	Q	$\overline{Q}$
Data Enabled	H	L	L	H
	H	H	H	L
Data Latched	L	X	q	$\overline{q}$

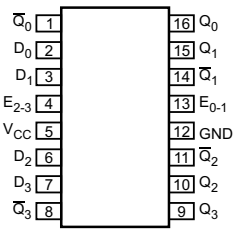
H = HIGH voltage level

L = LOW voltage level

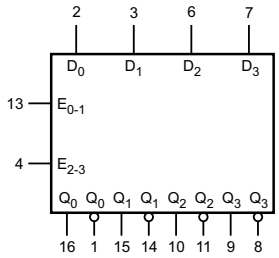
X = Don't care

q = Lower case letters indicate the state of referenced output one setup time prior to the

PIN CONFIGURATION

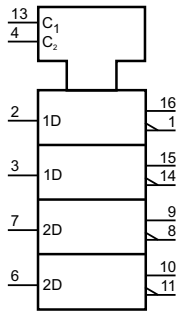


LOGIC SYMBOL



V_{CC} = Pin 5
GND = Pin 12

LOGIC SYMBOL (IEEE/IEC)



ADDERS 54/7483, LS83A

4-Bit Full Adder

- **High speed 4-bit binary addition**
- **Cascadeable in 4-bit increments**
- **LS83A has fast internal carry lookahead**
- **See' 283 for corner power pin version**

TYPE	TYPICAL ADD TIMES (Two 8-bit Words)	TYPICAL SUPPLY CURRENT (Total)
7483	23ns	66mA
74LS83A	25ns	19mA

DESCRIPTION

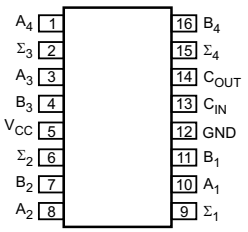
The '83 adds two 4-bit binary words (A_n plus B_n) plus the incoming carry. The binary sum appears on the Sum outputs ($\Sigma_1 - \Sigma_4$) and the outgoing carry (C_{OUT}) according to the equation.

$$C_{IN} + (A_1 + B_1) + 2(A_2 + B_2) + 4(A_3 + B_3) + 8(A_4 + B_4) = \Sigma_1 + 2\Sigma_2 + 4\Sigma_3 + 8\Sigma_4 + 16C_{OUT}$$

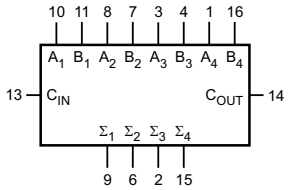
where (+) = plus.

Due to the symmetry of the binary add function, the '83 can be used with either all active-HIGH operands (positive logic) or with all active-LOW operands (negative logic). See Function Table. With active-HIGH inputs, C_{IN} cannot be left open; it must be held LOW when no "carry in" is intended. Interchanging inputs of equal weight does not affect the operation, thus C_{IN}, A_1, B_1, can arbitrarily be assigned to pins 10, 11, 13, etc.

PIN CONFIGURATION

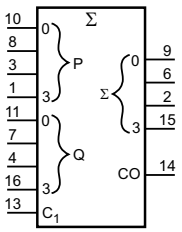


LOGIC SYMBOL



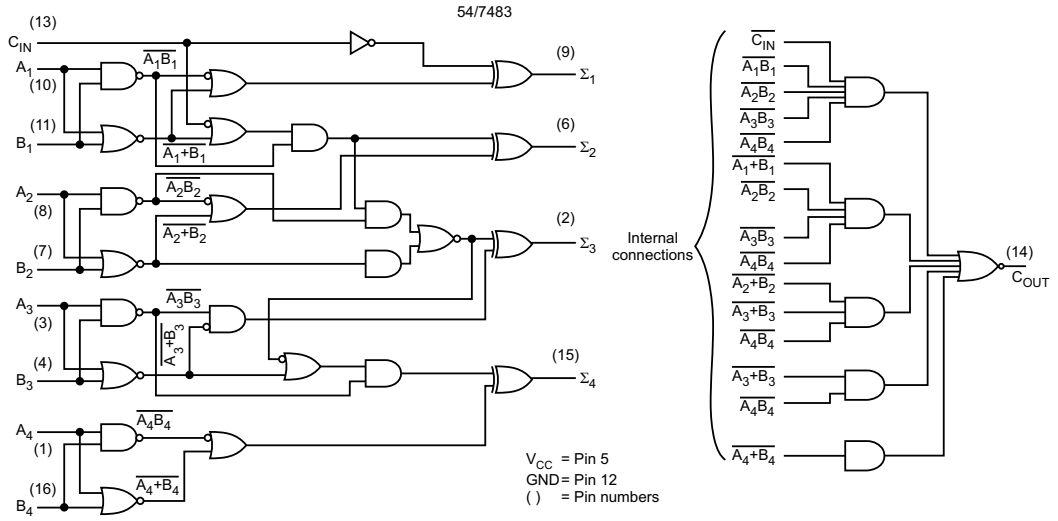
V_{CC} = Pin 5
GND = Pin 12

LOGIC SYMBOL (IEEE/IEC)

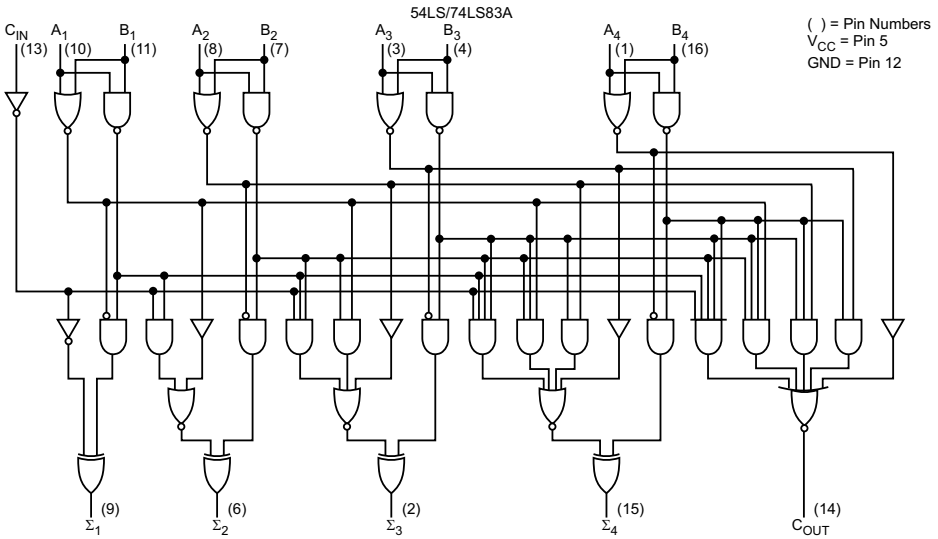


ADDERS 54/7483, LS83A

LOGIC DIAGRAM



LOGIC DIAGRAM



COMPARATORS 54/7485, LS85, S85

4-Bit Magnitude Comparator

- Magnitude comparison of any binary words
- Serial or parallel expansion without extra gating
- Use 54S/74S85 for very high speed comparisons

TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (Total)
7485	23ns	55mA
74LS85	23ns	10mA

DESCRIPTION

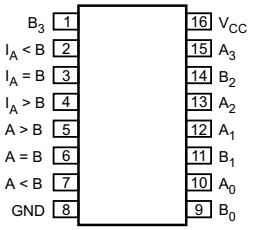
The '85 is a 4-bit magnitude comparator that can be expanded to almost any length. It compares two 4-bit binary, BCD, or other monotonic codes and presents the three possible magnitude results at the outputs. The 4-bit inputs are weighted ($A_0 - A_3$) and ($B_0 - B_3$), where A_3 and B_3 are the most significant bits.

The operation of the '85 is described in the Function Table, showing all possible logic conditions. The upper part of the table describes the normal operation under all conditions that will occur in a single device or in a series expansion scheme. In the upper part of the table the three outputs are mutually exclusive. in the lower part of the table, the outputs reflect the feed-forward conditions that exist in the parallel expansion schemes.

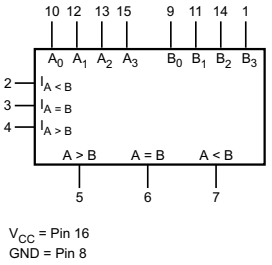
The expansion inputs $I_{A>B}$, $I_{A=B}$ and $I_{A<B}$ are the least significant bit positions. When used for series expansion, the A > B, A = B and A < B outputs of the least significant word are connected to the corresponding $I_{A>B}$, $I_{A=B}$ and $I_{A<B}$inputs of the next higher stage. Stages can be added in this manner to any length, but a propagation delay penalty of about 15ns is added with each additional stage. For proper operation the expansion inputs of the least significant word should be tied as follows : $I_{A>B}$ = LOW $I_{A=B}$ = HIGH and $I_{A<B}$ = LOW.

The parallel expansion scheme is the most efficient general use of these comparators, in the parallel expansion scheme, the expansion inputs can be used as a fifth input bit position except on the least significant device which must be connected as in the serial scheme. The expansion inputs are used by labeling $I_{A>B}$ as an "A" input, $I_{A<B}$ as a "B" input and setting $I_{A=B}$ LOW. The '85 can be used as a 5-bit comparator only when the outputs are used to drive the ($A_0 - A_3$) and ($B_0 - B_3$) inputs of another '85 device. The parallel technique can be expanded to any number of bits as shown in Table.

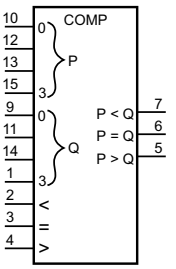
PIN CONFIGURATION



LOGIC SYMBOL

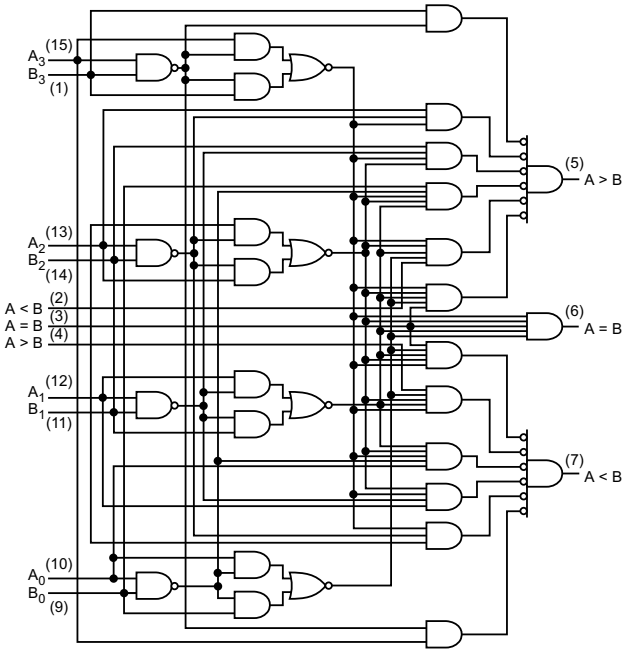


LOGIC SYMBOL (IEEE/IEC)



COMPARATORS 54/7485, LS85, S85

LOGIC DIAGRAM



FUNCTION TABLE

COMPARING INPUTS				CASCADING INPUTS			OUTPUTS		
A_3, B_3	A_2, B_2	A_1, B_1	A_0, B_0	$I_{A>B}$	$I_{A<B}$	$I_{A=B}$	A > B	A < B	A = B
$A_3 > B_3$	X	X	X	X	X	X	H	L	L
$A_3 < B_3$	X	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 > B_2$	X	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 < B_2$	X	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 > B_1$	X	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 < B_1$	X	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 > B_0$	X	X	X	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 < B_0$	X	X	X	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	L	L	H	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	H	L	L	H	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	X	X	H	L	L	H
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	H	H	L	L	L	L
$A_3 = B_3$	$A_2 = B_2$	$A_1 = B_1$	$A_0 = B_0$	L	L	L	H	H	L

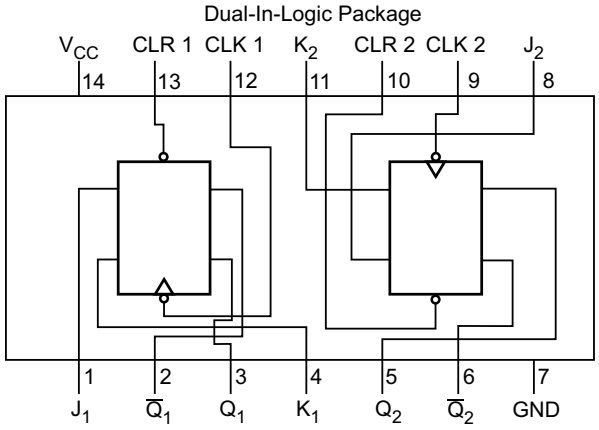
H = HIGH voltage level L = LOW voltage level X = Don't care

DM54LS107A / DM74LS107A Dual Negative-Edge-Triggered Master-Slave J-K Flip-Flops with Clear, and Complementary Outputs

GENERAL DESCRIPTION

This device contains two independent negative-edge-triggered J-K flip-flops with complementary outputs. The J and K data is processed by the flip-flops on the falling edge of the clock pulse. The clock triggering occurs at a voltage level and is not directly related to the transition time of the negative going edge of the clock pulse. The data on the J and K inputs may change while the clock is high or low without affecting the outputs as long as setup and hold times are not violated. A low logic level on the clear input will reset the outputs regardless of the logic levels of the other inputs.

CONNECTION DIAGRAM



FUNCTION TABLE

Inputs				Output	
CLR	CLK	J	K	Q	$\overline{Q}$
L	X	X	X	L	H
H	↓	L	L	Q_0	$\overline{Q}_0$
H	↓	H	L	H	L
H	↓	L	H	L	H
H	↓	H	H	Toggle	
H	H	X	X	Q_0	$\overline{Q}_0$

H = High Logic Level

X = Either Low or High Logic Level

L = Low Logic Leve

↓ = Negtive going-edge of pulse

Q_0 = The output logic level before the indicated input conditions were established

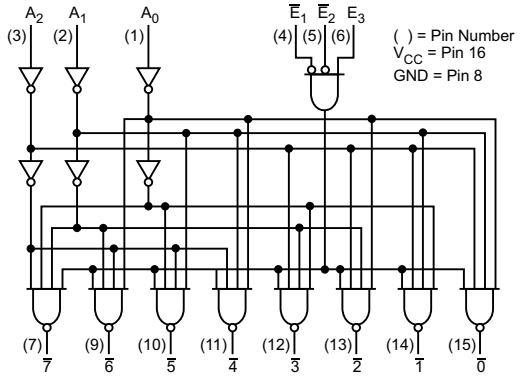
Toggle = Each output changes to the complement of its previous level on each falling edge of the clock pulse

DECODERS/DEMULTIPLEXERS 54/74LS138, S138

1-of-8 Decoder/Demultiplexer

- **Demultiplexing capability**
- **Multiple input enable for easy expansion**
- **Ideal for memory chip select decoding**
- **Direct replacement for Intel 3205**

LOGIC DIAGRAM



DESCRIPTION

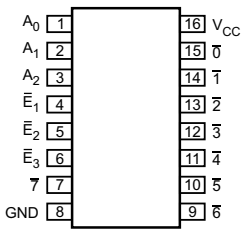
The '138 decoder accepts three binary weighted inputs (A_0, A_1, A_2) and when enabled, provides eight mutually exclusive, active LOW outputs ($\overline{0}-\overline{7}$). The device features three Enable inputs; two active LOW ($\overline{E}_1$, $\overline{E}_2$) and one active HIGH (E_3). Every output will be HIGH unless $\overline{E}_1$ and $\overline{E}_2$ are LOW and E_3 is HIGH. This multiple enable function allows easy parallel expansion of the device to a 1-of-32 (5 lines to 32 lines) decoder with just four '138s and one inverter.

The device can be used as an eight output demultiplexer by using one of the active LOW Enable inputs as the Data input and the remaining Enable inputs as strobes. Enable inputs not used must be permanently tied to their appropriate active HIGH or active LOW state.

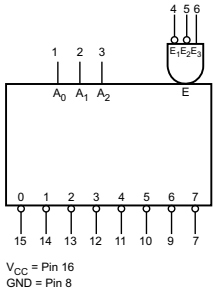
FUNCTION TABLE

INPUTS						OUTPUTS							
$\overline{E}_1$	$\overline{E}_2$	E_3	A_0	A_1	A_3	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$	$\overline{4}$	$\overline{5}$	$\overline{6}$	$\overline{7}$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

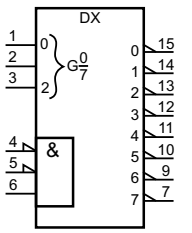
PIN CONFIGURATION



LOGIC SYMBOL



LOGIC SYMBOL (IEEE/IEC)



DECODERS/DEMULTIPLEXERS 54/74LS139, S139

Dual 1-or-4 Decoder/Demultiplexer

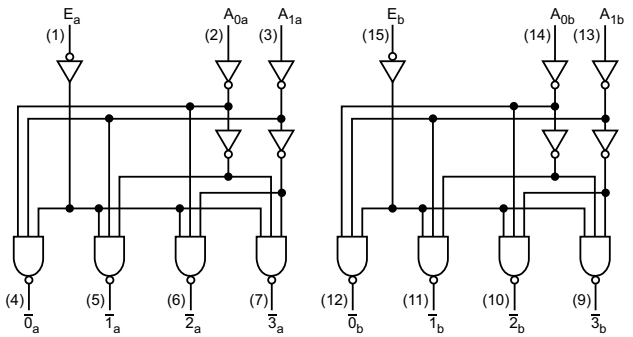
- **Demultiplexing capability**
- **Two independent 1-of-4 decoders**
- **Multifunction capability**
- **Replaces 9321 and 93L21 for higher performance**

TYPE	TYPICAL PROPAGATION DELAY (Enable at 2 logic levels)	TYPICAL SUPPLY CURRENT (Total)
74LS139	19ns	6.8mA
74S139	6ns	60mA

DESCRIPTION

The '139 is a high-speed, dual 1-of-4 decoder/demultiplexer. This device has two independent decoders, each accepting two binary weighted inputs (A_0, A_1) and providing four mutually exclusive active LOW outputs ($\overline{0} - \overline{3}$). Each decoder has an active LOW Enable ($\overline{E}$). When $\overline{E}$ is HIGH, every output is forced HIGH. The Enable can be used as the Data input for a 1-of-4.

LOGIC DIAGRAM

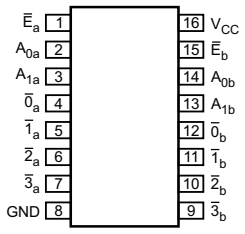


FUNCTION TABLE

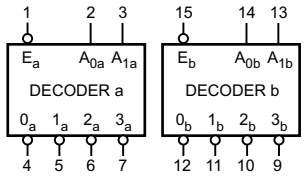
INPUTS			OUTPUTS			
$\overline{E}$	A_0	A_1	$\overline{0}$	$\overline{1}$	$\overline{2}$	$\overline{3}$
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	H	L	H	L	H	H
L	L	H	H	H	L	H
L	H	H	H	H	H	L

H = HIGH voltage level
L = LOW voltage level

PIN CONFIGURATION

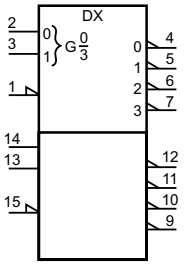


LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

LOGIC SYMBOL (IEEE/IEC)



ENCODER 54/74147

10-Line-To-4-Line Priority Encoder

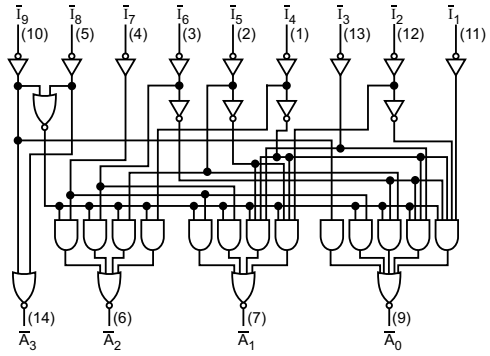
- **Encodes 10-line decimal to 4-line BCD.**
- **Useful for 10-position switch encoding.**
- **Used in code converters and generators.**

DESCRIPTION

The '147 9-input priority encoder accepts data from nine active-LOW inputs ($\bar{I}_1 - \bar{I}_9$) and provides a binary representation on the four active-LOW outputs ($A_0 - A_3$). A priority is assigned to each input so that when two or more inputs are simultaneously active, the input with the highest priority is represented on the output, with input line $\bar{I}_9$ having the highest priority.

The device provides the 10-line-to-4-line priority encoding function by use of the implied decimal "zero". The "zero" is encoded when all nine data inputs are HIGH, forcing all four outputs HIGH.

LOGIC DIAGRAM

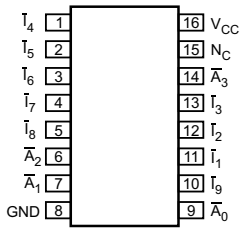


( ) = Pin Numbers
VCC = Pin 16
GND = Pin 8

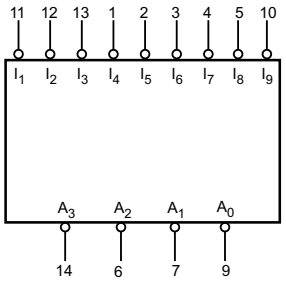
FUNCTION TABLE

INPUTS									OUTPUTS			
$\bar{I}_1$	$\bar{I}_2$	$\bar{I}_3$	$\bar{I}_4$	$\bar{I}_5$	$\bar{I}_6$	$\bar{I}_7$	$\bar{I}_8$	$\bar{I}_9$	$\bar{A}_3$	$\bar{A}_2$	$\bar{A}_1$	$\bar{A}_0$
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

PIN CONFIGURATION

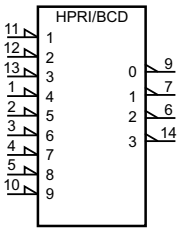


LOGIC SYMBOL



V_{CC} = Pin 16
GND = Pin 8

LOGIC SYMBOL (IEEE/IEC)



ENCODER 54/74148

8-Input Priority Encoder

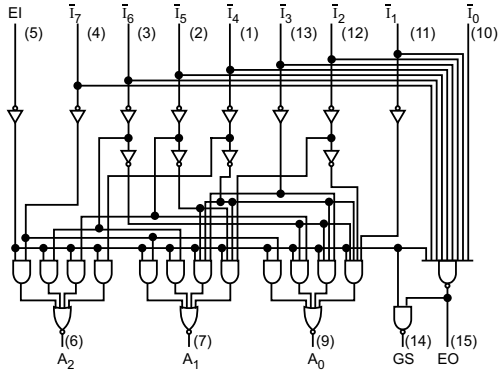
- **Code conversions.**
- **Multi-channel D/A converter.**
- **Decimal-to-BCD converter.**
- **Cascading for priority encoding of "N" bits.**
- **Input Enable capability.**
- **Priority encoding-automatic selection of highest priority input line.**
- **Output Enable-active LOW when all inputs HIGH.**
- **Group Signal output-active when any input is LOW.**

DESCRIPTION

The '148 8-input priority encoder accepts data from eight active-LOW inputs and provides a binary representation on the three active-LOW outputs. A priority is assigned to each input so that when two or more inputs are simultaneously active, the input with the highest priority is represented on the output, with input time $\overline{I}_7$ having the highest priority. A HIGH on the Enable Input ($\overline{EI}$) will force all outputs to the inactive (HIGH) state and allow new data to settle without producing erroneous information at the outputs.

A Group Signal ($\overline{GS}$) output and an Enable Output ($\overline{EO}$) are provided with the three data outputs. The $\overline{GS}$ is active-LOW when any input is LOW; this indicates when any input is active. The $\overline{EO}$ is active-LOW when all inputs are HIGH. Using the Enable Output along with the Enable Input allows priority encoding of N input signals. Both $\overline{EO}$ and $\overline{GS}$ are active-HIGH when the Enable Input is HIGH.

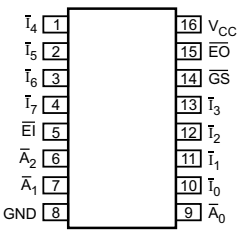
LOGIC DIAGRAM



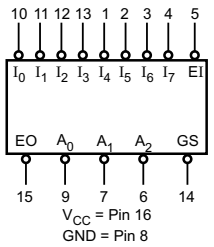
FUNCTION TABLE

INPUTS									OUTPUTS				
$\overline{EI}$	$\overline{I}_0$	$\overline{I}_1$	$\overline{I}_2$	$\overline{I}_3$	$\overline{I}_4$	$\overline{I}_5$	$\overline{I}_6$	$\overline{I}_7$	$\overline{GS}$	$\overline{A}_0$	$\overline{A}_1$	$\overline{A}_2$	$\overline{EO}$
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	H	L	L	H
L	X	X	X	X	X	L	H	H	L	L	H	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	L	L	L	H	H
L	X	X	L	H	H	H	H	H	L	H	L	H	H
L	X	L	H	H	H	H	H	H	L	L	H	H	H
L	L	H	H	H	H	H	H	H	L	H	H	H	H

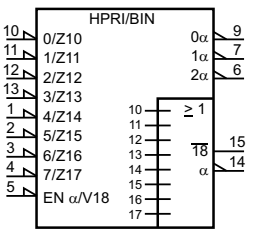
PIN CONFIGURATION



LOGIC SYMBOL



LOGIC SYMBOL (IEEE/IEC)



MULTIPLEXER 54/74150

16-Input Multiplexer

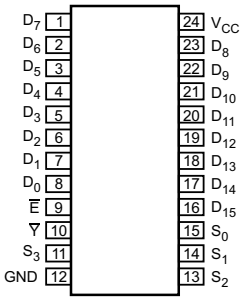
- **Select data from 16 sources**
- **Demultiplexing capability**
- **Active-LOW enable or strobe**
- **Inverting data output**

TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (Total)
74150	17ns	40mA

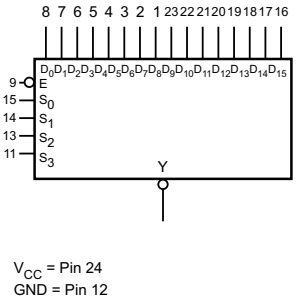
DESCRIPTION

The '150 is a logical implementation of a single-pole, 16-position switch with the switch position controlled by the state of four Select inputs, S_0, S_1, S_2, S_3. The Multiplexer output ($\overline{Y}$) inverts the selected data. The Enable input ($\overline{E}$) is active-LOW. When $\overline{E}$ is HIGH the $\overline{Y}$ output is HIGH regardless of all other inputs in one package the '150 provides the ability to select from 16 sources of data or control information.

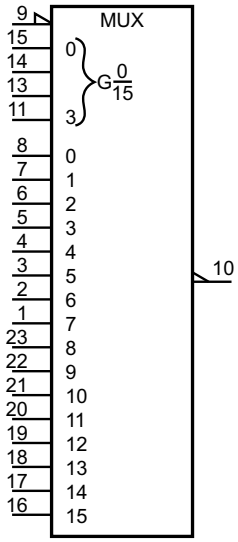
PIN CONFIGURATION



LOGIC SYMBOL

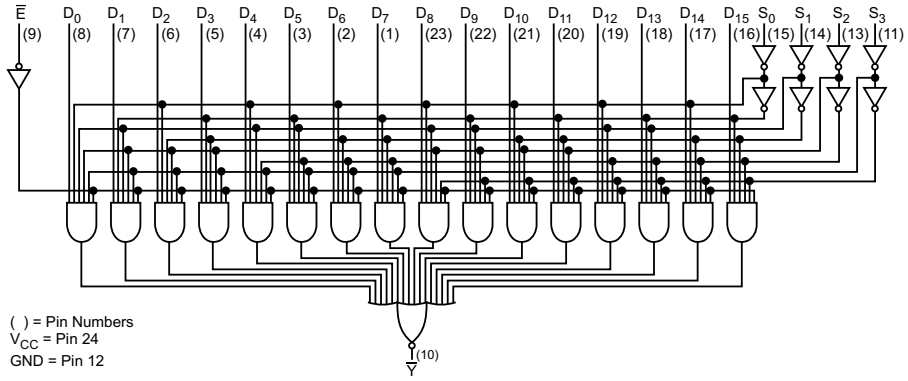


LOGIC SYMBOL (IEEE/IEC)



MULTIPLEXER 54/74150

LOGIC DIAGRAM



FUNCTION TABLE

INPUTS																			OUTPUT
S_3	S_2	S_1	S_0	$\overline{E}$	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}	D_{13}	D_{14}	D_{15}	$\overline{Y}$
X	X	X	X	H	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	H
L	L	L	L	L	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	H
L	L	L	L	L	H	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L
L	L	L	H	L	X	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	H
L	L	L	H	L	X	H	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L
L	L	H	L	L	X	X	L	X	X	X	X	X	X	X	X	X	X	X	X	X	H
L	L	H	L	L	X	X	H	X	X	X	X	X	X	X	X	X	X	X	X	X	L
L	L	H	H	L	X	X	X	L	X	X	X	X	X	X	X	X	X	X	X	X	H
L	L	H	H	L	X	X	X	H	X	X	X	X	X	X	X	X	X	X	X	X	L
L	H	L	L	L	X	X	X	X	L	X	X	X	X	X	X	X	X	X	X	X	H
L	H	L	L	L	X	X	X	X	H	X	X	X	X	X	X	X	X	X	X	X	L
L	H	L	H	L	X	X	X	X	X	L	X	X	X	X	X	X	X	X	X	X	H
L	H	L	H	L	X	X	X	X	X	H	X	X	X	X	X	X	X	X	X	X	L
L	H	H	L	L	X	X	X	X	X	X	L	X	X	X	X	X	X	X	X	X	H
L	H	H	L	L	X	X	X	X	X	X	H	X	X	X	X	X	X	X	X	X	L
L	H	H	H	L	X	X	X	X	X	X	X	L	X	X	X	X	X	X	X	X	H
L	H	H	H	L	X	X	X	X	X	X	X	H	X	X	X	X	X	X	X	X	L
H	L	L	L	L	X	X	X	X	X	X	X	X	L	X	X	X	X	X	X	X	H
H	L	L	L	L	X	X	X	X	X	X	X	X	H	X	X	X	X	X	X	X	L
H	L	L	H	L	X	X	X	X	X	X	X	X	X	L	X	X	X	X	X	X	H
H	L	L	H	L	X	X	X	X	X	X	X	X	X	H	X	X	X	X	X	X	L
H	L	H	L	L	X	X	X	X	X	X	X	X	X	X	L	X	X	X	X	X	H
H	L	H	L	L	X	X	X	X	X	X	X	X	X	X	H	X	X	X	X	X	L
H	L	H	H	L	X	X	X	X	X	X	X	X	X	X	X	L	X	X	X	X	H
H	L	H	H	L	X	X	X	X	X	X	X	X	X	X	X	H	X	X	X	X	L
H	H	L	L	L	X	X	X	X	X	X	X	X	X	X	X	X	L	X	X	X	H
H	H	L	L	L	X	X	X	X	X	X	X	X	X	X	X	X	H	X	X	X	L
H	H	L	H	L	X	X	X	X	X	X	X	X	X	X	X	X	X	L	X	X	H
H	H	L	H	L	X	X	X	X	X	X	X	X	X	X	X	X	X	H	X	X	L
H	H	H	L	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	X	H
H	H	H	L	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	H	X	L
H	H	H	H	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	L	H
H	H	H	H	L	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	H	L

H = HIGH voltage level L = LOW voltage level X = Don't care

MULTIPLEXERS 54/74151, LS151, S151

8-Input Multiplexer

- **Multifunction capability**
- **Complementary outputs**
- **See '251 for 3-state version**

TYPE	TYPICAL PROPAGATION DELAY (Enable to $\overline{Y}$)	TYPICAL SUPPLY CURRENT (Total)
74151	18ns	29mA
74LS151	12ns	6mA

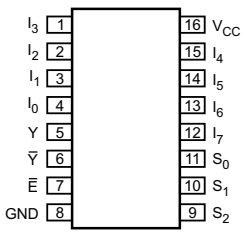
DESCRIPTION

The '151 is a logical implementation of a single-pole, 8-position switch with the switch position controlled by the state of three Select inputs, S_0, S_1, S_2. True (Y) and Complement ($\overline{Y}$) outputs are both provided. The Enable input ($\overline{E}$) is active LOW when $\overline{E}$ is HIGH, the $\overline{Y}$ output is HIGH and the Y output is LOW. regardless of all other inputs. The logic function provided at the output is :

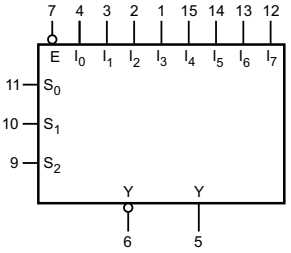
$$\begin{aligned} Y = \; & \overline{E}\cdot(I_0\cdot\overline{S}_0\cdot\overline{S}_1\cdot\overline{S}_2 + I_1\cdot S_0\cdot\overline{S}_1\cdot\overline{S}_2 \\ & + I_2\cdot\overline{S}_0\cdot S_1\cdot\overline{S}_2 + I_3\cdot S_0\cdot S_1\cdot\overline{S}_2 \\ & + I_4\cdot\overline{S}_0\cdot\overline{S}_1\cdot S_2 + I_5\cdot S_0\cdot\overline{S}_1\cdot S_2 \\ & + I_6\cdot\overline{S}_0\cdot S_1\cdot S_2 + I_7\cdot S_0\cdot S_1\cdot S_2) \end{aligned}$$

In one package the '151 provides the ability to select from eight sources of data or control information. The device can provide any logic function of four variables and its negation with correct manipulation.

PIN CONFIGURATION

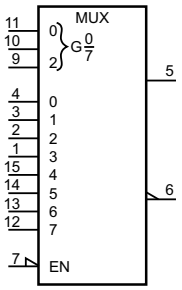


LOGIC SYMBOL



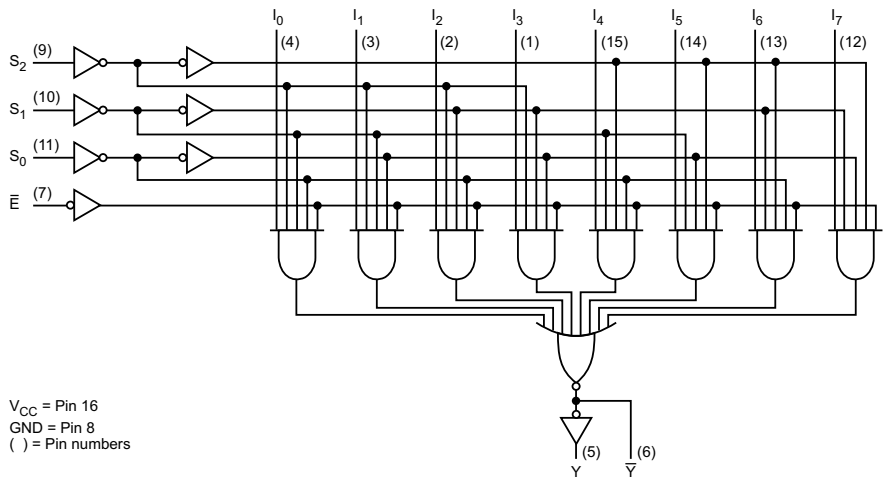
V_{CC} = Pin 16
GND = Pin 8

LOGIC SYMBOL (IEEE/IEC)



MULTIPLEXERS 54/74151, LS151, S151

LOGIC DIAGRAM



FUNCTION TABLE

				INPUTS								OUTPUT	
$\overline{E}$	S_2	S_1	S_0	I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$\overline{Y}$	Y
H	X	X	X	X	X	X	X	X	X	X	X	H	L
L	L	L	L	L	X	X	X	X	X	X	X	H	L
L	L	L	L	H	X	X	X	X	X	X	X	L	H
L	L	L	H	X	L	X	X	X	X	X	X	H	L
L	L	L	H	X	H	X	X	X	X	X	X	L	H
L	L	H	L	X	X	L	X	X	X	X	X	H	L
L	L	H	L	X	X	H	X	X	X	X	X	L	H
L	L	H	H	X	X	X	L	X	X	X	X	H	L
L	L	H	H	X	X	X	H	X	X	X	X	L	H
L	H	L	L	X	X	X	X	L	X	X	X	H	L
L	H	L	L	X	X	X	X	H	X	X	X	L	H
L	H	L	H	X	X	X	X	X	L	X	X	H	L
L	H	L	H	X	X	X	X	X	H	X	X	L	H
L	H	H	L	X	X	X	X	X	X	L	X	H	L
L	H	H	L	X	X	X	X	X	X	H	X	L	H
L	H	H	H	X	X	X	X	X	X	X	L	H	L
L	H	H	H	X	X	X	X	X	X	X	H	L	H

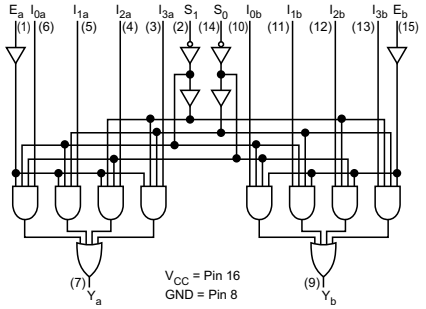
H = HIGH voltage level L = LOW voltage level X = Don't care

MULTIPLEXERS 54/74153, LS153, S153

Dual 4-Line to 1-Line Multiplexer

- **Non-inverting outputs**
- **Separate Enable for each section**
- **Common Select inputs**
- **See '253 for 3-State version**

LOGIC DIAGRAM



DESCRIPTION

The '153 is a dual 4-input multiplexer that can select 2 bits of data from up to four sources under control of the common Select inputs (S_0, S_1). The two 4-input multiplexer circuits have individual active LOW Enables ($\overline{E}_a$, $\overline{E}_b$) which can be used to strobe the outputs independently. Outputs (Y_a, Y_b) are forced LOW when the corresponding Enables ($\overline{E}_a$, $\overline{E}_b$) are HIGH.

The device is the logical implementation of a 2-pole, 4-position switch where the position of the switch is determined by the logic levels supplied to the two select inputs. The logic equations for the outputs are shown below.

$$Y_a = \overline{E}_a \cdot (I_{0a} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{1a} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{2a} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{3a} \cdot \overline{S}_1 \cdot \overline{S}_0)$$

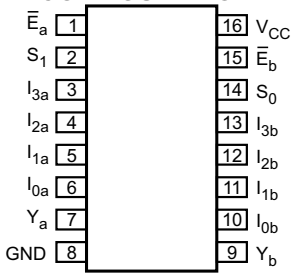
$$Y_b = \overline{E}_b \cdot (I_{0b} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{1b} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{2b} \cdot \overline{S}_1 \cdot \overline{S}_0 + I_{3b} \cdot \overline{S}_1 \cdot \overline{S}_0)$$

FUNCTION TABLE

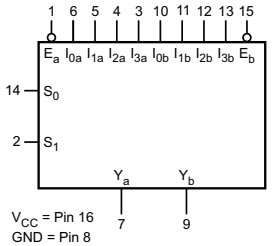
SELECT INPUTS		INPUTS (a or b)					OUT-PUT
S_0	S_1	$\overline{E}$	I_0	I_1	I_2	I_3	Y
X	X	H	X	X	X	X	L
L	L	L	L	X	X	X	L
L	L	L	H	X	X	X	H
H	L	L	X	L	X	X	L
H	L	L	X	H	X	X	H
L	H	L	X	X	L	X	L
L	H	L	X	X	H	X	H
H	H	L	X	X	X	L	L
H	H	L	X	X	X	H	H

The '153 can be used to move data to a common output bus from a group of registers. The state of the Select inputs would determine the particular register from which the data came. An alternative application is as a function generator. The device can generate two functions or three variables. This is useful for implementing highly irregular random logic.

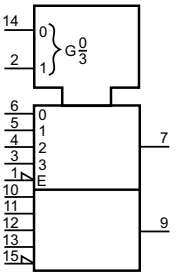
PIN CONFIGURATION



LOGIC SYMBOL



LOGIC SYMBOL (IEEE/IEC)



Other Popular Books by the Same Author

1. Advance Processors (8086, 80186/88, 80286, 80386, 80486 and Pentium Processors)
2. Communication Engineering
3. Computer Architecture & Organisation
4. Computer Graphics
5. Computer Graphics & Multimedia
6. Computer Programming using C
7. Computer System and Programming in C
8. Digital Electronics
9. Digital IC Applications
10. Digital Logic Circuits using VHDL
11. Digital Principles and Systems Design using Verilog HDL
12. Electronic Circuits I
13. Electronic Circuits II
14. Electronic Devices and Circuits - I
15. Electronic Devices and Circuits - II
16. Electronics & Microprocessor (Electronic Devices, Introduction to Digital Electronics and 8085 Microprocessor)
17. Embedded Systems
18. Fundamentals of HDL
19. Linear & Digital IC Applications
20. Linear Integrated Circuits & Applications
21. Mechatronics & Microprocessor
22. Microcontrollers (8051 and MSP430 Microcontrollers Family - Architecture & Programming)
23. Microprocessor & Multicore Systems
24. Microprocessor and Control Systems
25. Microprocessor Techniques (8085 and 8086 - Architecture, Instruction set, Programming and Peripheral Interfacing)
26. Microprocessors and Introduction to Microcontroller (8085, 8086, 8051 - Architecture, Interfacing and Programming)
27. Microprocessors and Microcontrollers (8086, 8051 - Architecture, Interfacing and Programming)
28. Microprocessors and Microcontrollers Interfacing Techniques
29. Pulse and Digital Circuits

ISBN 978-93-5099-232-6

TECHNICAL PUBLICATIONS™
An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth, Pune - 411030, M.S. INDIA.
Ph.: +91-020-24495496/97, Telefax : +91-020-24495497
Email : technical@vtubooks.com Website : www.vtubooks.com
eBooks available on www.technicalpublications.org